Multi Agent System Summary

Author: **Nicolò Brandizzi**

Contributors:

SAPIENZA
UNIVERSITÀ DI ROMA

DIAG
Sapienza
November 2018

Contents

Abstract

This is **free** material! You should not spend money on it.
This notes are about the *Multi Agent System* part taught by professor Daniele Nardi in the Artificial Intelligence class. Everyone is welcome to contribute to this notes in any relevant form, just ask for a pull request and be patient.
Remember to add your name under the contributors list in the title page when submitting some changes (if you feel like it).

Contents

1 Adversarial Games

1.1 Games

In AI a common game has the following proprieties:

- **Zero-sum**: is when the total payoff to all player is the same for every instance of the game (better called *constant-sum*) [1].
- **Perfect Information**: everything in the game is visible [2].
- Multi-agent with *equal and opposite utility values* [3].

Moreover a game is composed of the following elements:

- Initial state
- *Players(s)*: which player has to move in a state.
- *Actions(s)*: return set of legal states.
- *Result(s,a)*: defines the result of an action (transition-model).
- *Terminal-Test(s)*: true when the game is over, false otherwise.
- *Utility(s,p)*: defines the final numeric value for a game that ends in the *terminal state* s for a player p.

Given the initial player MAX and the other player MIN, we can look at a game as a tree where each node is a state and the edges are moves, by doing so it is easy to get how the player MAX generates an OR node while the player MIN generates an AND node [4]. This is because MAX do not know which will be the action chosen by MIN so it has to build a *contingent plan* in order to consider every possible outcome.

The **MinMax strategy** is a strategy with the following heuristic:

$$MinMax(s) = \begin{cases} Utility(s) & \text{if } TerminalTest(s) \\ \max_{a \in Actions(s)} MinMax(Result(s,a)) & \text{if } Playes(s) = Max \\ \min_{a \in Actions(s)} MinMax(Result(s,a)) & \text{if } Playes(s) = Min \end{cases}$$

That is the minmax value of a node is the utility (for MAX) of being in that state, assuming that both players play optimally [5].

[1] An example is chess where you can either win +1, loose -1 or draw 1/2 so that the sum of two player is always 1.

[2] Not like games like poker in which not all the cards are visible.

[3] One player wins the other looses (adversarial).

[4] Just like in the non-deterministic search problem, we don't know what MIN action will be as we do not know if a non action has occurred in a sensor-less environment.

[5] If MIN is not optimal than MAX will do even better.

MinMax Algorithm It make use of a recursive computation of the minmax values of each successor state, than the values are backed up through the tree as the recursion unwinds. It is like a depth-first exploration where the maximum dept of the tree is m and there are b legal moves.

- **Complete**: yes, if $m < \infty$
- **Optimal**: yes, if MIN is optimal.
- **Time complexity**: $O(b^m)$
- **Space complexity**: $O(b \cdot m)$

Multiplayer games Instead of a value for each node there is a vector, one value for each agent in the game.
Alliances may arise from purely selfish behavior.

1.2 Alpha-Beta Pruning

It is used to prune away branches that cannot influence the final decision. The logic is pretty simple:
Consider a node n (in the figure it is the node C) somewhere in the tree, such that MAX can choose to move to that node (since MAX starts form A he can choose between either [B,C,D]). If there is a better choice m at n parent or any point up (in this case the better move would be node B since MIN will chose the minimum value which is 3, if C is considered then MIN could chose $2 < 3$), then n will never be reached, hence it can be pruned.

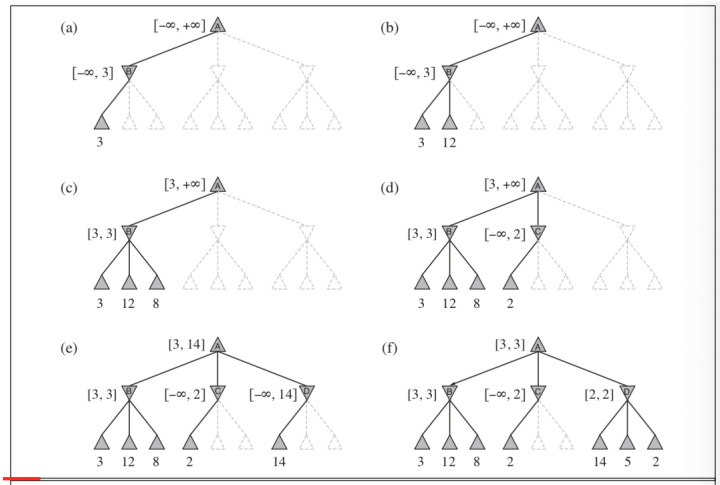


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successor states, so the value of B is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

The name alpha-beta derives from the bipolarity of the standard two player game:

- α is the value of the best choice for MAX, i.e. highest value
- β is the value of the best choice for MIN, i.e. lowest value

Each one comes after the other with alpha starting.

Move Ordering since alpha-beta pruning is dependent on the order in which the states are examined it may be convenient to order them from the lowest to the highest, thus pruning a large part of the search space theoretically bringing the time complexity to $O(b^{m/2})$. On the other hand by examining the successors in a random order we can get a time complexity of $O(b^{3m/4})$.
We can use **dynamic move-ordering schemes**, that first try the moves that were found out to be the best in the past.
A **transposition table** is a way to store the evaluation of the resulting position in a hash table, so that we do not have to recompute it on subsequent occurrence.

1.3 Imperfect Real-Time Decisions

We can use an evaluation function as the cutoff value for iterative deepening instead of a fixed depth d. This will turn non-terminal nodes into terminal leaves. This can be easily done by slightly modified the minmax algorithm including a heuristic function EVAL which estimates the position utility and replace the terminal test with a cutoff test that decides when to use EVAL.

Evaluation Functions How do we design a good evaluation function?

1. The evaluation function should order the terminal states in the same way as the true utility function ($win > draw > loss$).
2. Computation must not take too long.
3. The evaluation function should be correlated with the actual chances of winning.

How can we get a correlation between estimated and real evaluation functions?

We can have features that taken together defines *equivalence classes* of states: the states in each category have the same values for all the features.
The evaluation function can return a single value that reflect the proportion of states with each other, for example:
We know that, in chess, when we encounter the two-pawns vs. one-pawn category there is a 0.72 chance of winning (+1), 0.2 of drawing (0), 0.08 of loosing (2), so we can use the **expected value**:

$$(0.72 \cdot 1) + (0.2 \cdot 0) + (0.08 \cdot +1/2) = 0.76$$

This kind of evaluation function is called **weighted linear function** because it can be expressed as:

$$Eval(s) = w_1 f_1(s) + w_2 + f_2(s) + ... + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

where w_i is a weight and f_i is a feature [6].

Cutting off search We can use incremental depth on each recursive call with iterative deepening, using a *CuttOff-Test(state,depth)* function. When the time runs out the program return the move selected by the deepest complete search.
There are some cases in which the approximated nature of the evaluation function has to be addressed, we introduce the notion of **quiescent** position, which are those position that are unlikely to exhibit wild swings in value in the near future. With this we ca use a **quiescent search**.
Another problem is the one of the **horizon effect** in which the program is delaying an unavoidable move which will deal serious damage, doing so it can choose some bad moves which will be worst as soon as the unavoidable action takes effect. We can mitigate this behavior using **singular extension**, a move that is *clearly better* than all other moves in a given position is saved for later search.

Forward pruning This technique mimic the human brain that cannot process so many possibilities at once, thus pruning some moves at a given node immediately without further consideration. Local **beam search** can be used where the beam is the set of n best moves.
We can use the **ProbCut** function, which is a kind of forward pruning version of alpha-beta search that uses statistic generated from prior experiences.

[6] There can also be a non linear combination of features.

1.4 Stochastic Games

In stochastic games there is a random element which cannot be consider a priori. This leads to the creation of so called **chance nodes** in addition to AND, OR ones.
We use a **expect-minmax value**, in which for each position we calculate the *expected value* with the average over all possible outcomes.

2 Communication

The difference between agents and object oriented programs, is that agents can neither force other agents to perform some action, nor write data onto the internal state of other agents, thus the communication is an **action** that attempts to influence other agents.

2.1 Speech Acts

In *Speech theory*, speech is an action just like any other.

Austin Has been the first to study the influence of speech in the environment. He identified a number of *performative verbs* which corresponds to different types of speech acts such as *requests, inform and premise*.

Searle Given a SPEAKER who request a HEARER to perform an ACTION, the following proprieties must hold:

- **Normal I/O conditions**, where the HEARER is able to hear and the circumstance is normal (not in a movie).
- **Preparatory conditions**: HEARER must be able to perform ACTION and SPEAKER believes that HEARER is able to perform ACTION.
- **Sincerity conditions**: request must be sincere

There is also a division in category for the speech acts:

- **Representative**: informing.
- **Directives**: requesting (do something).
- **Commissives**: promising.
- **Expressives**: thanking (express psychological state).
- **Declarations** declare war (effect some changes).

Plan-based theory for speech acts There are two kind of precondition for a speech act (such as the *Request* act):

- *CanDo* preconditions (cando.pr): The SPEAKER believes that the HEARER is able to do the ACTION
- *Want* preconditions (want.pr): The SPEAKER must want the action to be performed.

The effect of this *Request* action is that the agent will come to believe it wants to do something if it believes that another agent believes it wants to do it. This is also the definition of *CasueToWant*.
On the other hand the *Inform* action has to goal for a SPEAKER to get a HEARER to believe some statement; the cando.pr is that the SPEAKER must believe the information ϕ and it believes that the hearer must believe that the SPEAKER believe ϕ.

2.2 Agent Communication Languages

KIF Stands for *Knowledge Interchange Format* is a communication language for expressing proprieties of a particular domain, i.e. message *content* [7]. It it based on first-order-logic and can express the following:

- properties of things in a domain (e.g. 'Michael is a vegetarian')
- relationships between things in a domain (e.g. 'Michael and Janine are married')
- general properties of a domain (e.g. 'everybody has a mother')

The kind of quantifiers it can use are *and, or, not, forall, exists.*

KQML Is a message-based language for agent communication where each message has a **performative** (the class of a message) and some **parameters**; here is an example:
(ask-one
:content (PRICE IBM ?price)
: stop-server
:language LPROLOG
s:ontology NYSE-TICKS
)

Where the class is *ask-one*, and each parameter has the following meaning

Parameter	Meaning
`:content`	content of the message
`:force`	whether the sender of the message will ever deny the content of the message
`:reply-with`	whether the sender expects a reply, and, if so, an identifier for the reply
`:in-reply-to`	reference to the `:reply-with` parameter
`:sender`	sender of the message
`:receiver`	intended recipient of the message

FIPA Stands for *Foundation for Intelligent Physical Agents* developed a stander for multi-agent communication with *20 performatives* shown below

[7] Not messages themselves.

Performative	Passing information	Requesting information	Negotiation	Performing actions	Error handling
accept-proposal			×		
agree				×	
cancel		×		×	
cfp			×		
confirm	×				
disconfirm	×				
failure					×
inform	×				
inform-if	×				
inform-ref	×				
not-understood					×
propagate				×	
propose			×		
proxy				×	
query-if		×			
query-ref		×			
refuse				×	
reject-proposal			×		
request				×	
request-when				×	
request-whenever				×	
subscribe		×			

Where :

- *cfp* is *call for proposal*
- *inform-if* can be translated with "tell me if the content of the inform-if is either true or false"
- *inform-ref* same as above but with values

The primitives in the FIPA language are:

- *Inform(i,j,ϕ)*: where agent i inform agent j about ϕ. Which have as effect the fact that j now believes ϕ ($B_j\phi$) and as preconditions $B_i\phi \wedge \neg B_i(Bif_i\phi \vee Uif_i\phi)$ where:
  - $B_i\phi$ means *agent i believes ϕ.*
  - $Bif_i\phi$ means *agent i has a **definitive** opinion on ϕ.*
  - $Uif_i\phi$ means *agent i is uncertain about ϕ.*
  - thus $\neg B_i(Bif_i\phi \vee Uif_i\phi)$ means that *agent i does not have a definite opinion on ϕ nor it is uncertain about it*, he simply believes it.
- *Request(i,j,α)*: where agent i wants agent j to perform an action α. This have as effect that action α is now executed and, as preconditions, we have $B_i Agent(\alpha, j) \wedge \neg B_i I_j Done(\alpha)$ where:
  - $B_i Agent(\alpha, j)$ means that j is the agent which can perform α.
  - $B_i I_j Done(\alpha)$ means that i believes that j does not intend to do action α yet.

There is also a **semantic conformance testing** in which we need to check both the syntax used (easy) and the semantic.

Ontologies for Agent Communication An ontology is a formal definition of a body of knowledge, the issue with ontologies arises when two agents which are communication about some domain need to agree on the terminology that they use to describe this domain.

3 Collaboration and Task Distribution

When we are working with multiple agents that have to share resources, help each other and other form of cooperation to solve a task, a question arises: *who has to do what? using what resources? Basing on what skills?*

3.1 Modes of task allocation

When we encounter tasks which requires a high level of resources [8], we need to brake down those tasks into sub-tasks and distribute them among various agents.

Working together There can either be a *benevolent scenario*, in which agents help each other by design, or a *self-interested* one in which both conflicts and agreements can arise.

Result sharing In this form of cooperation information relevant to the task are shared among agents with different kind of communication models (blackboard, publisher/subscriber).

Coordination Missing

Multiagent planning Missing

Criteria First we need to make sub-tasks as independent as possible to avoid redundant actions and high level of coordination.

Roles Then we split the agents into: *clients* which need something [9] and *supplies or servers* which are capable of supplying a service.

Forms of allocation There are various forms of tasks allocation:

- **Centralized allocation**, which can be divided into:
  - *Hierarchical structures*: where a superior agents order others around. It is a rigid allocation and the calls are specific of imperative languages.
  - *Egalitarian structures*: where there are some special agents, like **traders**, which manage all the allocation procedures.
- **Distributed allocation**: in which each agents try to obtain services from the suppliers by their own, there are two types:
  - *Acquaintance networks*: where each agent has a representation of other agents skills.

[8] Task which cannot be completed by a single agent for various reasons (lack of resources or knowledge...).

[9] Being information or work-force.

- *Contract networks*: where agents ask the others if they are capable/want to accept the task [10].

- **Emergent Allocation**: is a reactive system in which communication takes place as a reaction to stimuli[11] .

3.2 Centralized allocation by trader

It uses a acquaintance table indicating all the agents capable of executing task T. The algorithm works as follows:

1. When an agent X need to carry out a task T which cannot [12] do by itself it asks a **trader** agent to find an agent who can.
2. The trader then asks all the agents in the table which can execute task T, called suppliers.
3. The suppliers can either accept or refuse T .
4. X will be informed if any supplier is available for T.

Optimization It is possible to improve this mechanism in several ways by prioritizing the selection by most skilled agents:

- If the trader knows the evaluation function used by the suppliers then it can sort the best agents and return the first.
- Else it should first ask each supplier its skill for the specific task and then evaluate them.

We can move from a sequential execution, where the trader ask each supplier sequentially before passing on to the next, to a *parallel execution* by stopping the search for suppliers as long as one has responded.

Cons of Centralized Allocation The major problem of centralized allocation are:

- *Bottlenecks*, since each query must pass through a trader. moreover the number of messages reach $\alpha k N(2 + 2\beta N) \Rightarrow O(N^2)$, where:
  - $\alpha = PotentialClient/TotalClients$
  - $\beta = PotentialSuppliers/TotalSuppliers$
  - k =requests per time unity
- *Sensitivity to failures*, if the trader goes down so does the entire system.

[10] Dynamic and easy to implement, whereas the acquaintance is static.
[11] Emergent allocation will be treated in the Swarm robotic Section 6
[12] Or does not want to.

3.3 Acquaintance network (Distributed)

Assume that each agents has a the skill table [13] for the agents it *knows*. The skills tables are **correct** and **static**, but can be **partial** since we are working under the *partial representation system for multi-agents* hypothesis. Moreover the tables will not be updated so that if an agent experience failure or exit the system the trader will not know.
We can represent the table in form of matrices, where agents are columns and skills are rows.

	A	B	C	D
C1	0	1	1	0
C2	0	0	1	0
C3	1	0	0	1

or in a graph form in which agents are nodes and skills are arches.
Finally there are two types of allocation depending on whenever the agents can delegate their request to other agents or not.

Direct Allocation In this mode an agent can have a task executed only by an agent that it knows directly [14]. If agent X wants task T to be executed it asks every agents it knows until someone respond positively.
We can address the issue of not finding any supplier by forcing allocation or using a centralized system which:

- Plays the role of maintenance and repair agent.
- Will experience a low number of messages, since it is called only for extreme situations.
- It can update the agents acquaintance tables in order to accelerate allocation process.

Allocation by delegation With this technique we can link together clients and suppliers which do not know each other directly. In this allocation mode a supplier that is asked to carry out a task can send it to another agent if it is not capable of doing it.
Parallel searching and diffusion algorithm [15] can be used to propagate requests to all known agents. The are are some problems:

- Asking the same agent multiple time; can be fixed by marking the agent for that specific search [16].
- We want to make sure to search through the entire tree so we need acknowledges from every agent.

Finally there are some imperfections:

[13] Often represented in dictionary form.
[14] Similar to a single trader distribution system.
[15] Like parallel breadth search.
[16] Problem arises to make skill search unique.

- There is no optimization in term of agents skillfulness for a specific task.
- Stop the search as soon as an agent has accepted the task
- Unique task search key to avoid agent ignoring similar searches.

Reorganization of acquaintance network We can have environments in which the agents capabilities changes through time [17] so we need to reorganize the acquaintance network. We can do it in two ways:

- When the agent A changes then it alert all the other agents known to A. Two problems:
  - Need of bilateral contacts between acquaintance [18].
  - Parallelism synchronization between changes and new requests.
- When B asks A to execute a Task which is no longer capable of doing it can update its table. In this case the frequencies of changes becomes critical.

3.4 Allocation by contract net (Distributed)

The contract net is a task allocation mechanism based on a market-like protocol. The relationship between the *manager* (client) and the *bidders* (suppliers) is channeled through a request for bids [19] submitted by the bidders.
It works as follows:

1. **Announcement**: the manager sends the description of the task to all those considered able to respond or to everyone.
2. **Bidding**: the bidders submit *proposals* to the manager.
3. **Awarding**: the manager estimates the proposals and awards the contract to the best bidder.
4. **Expediting**: finally, the bidder awarded with the contract accepts or decline the task.

Contract propagation language The request for bids includes:

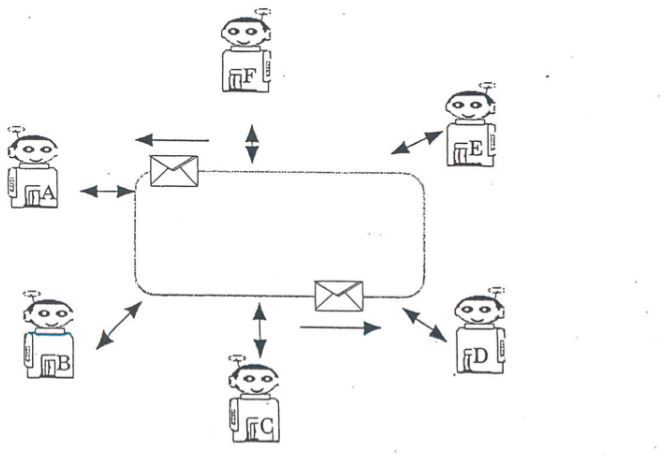
- Description of the task
- Contract number
- Definition of the qualities required for the task
- Form of proposal
- Expiry date

[17] Either agents enter/exit the system or their skills changes.
[18] If B is an acquaintance of A then A have to know its skills from B prospective.
[19] Bids are offers.

like in the following image:

```
Message: RequestForBids
  To: *
  From: A21
  DescriptionOfTask:
    TypeOfTask: check-feature
  QualitiesRequired :
    MustHave: Camera
  FormOfProposal:
    Position : (X,Y)
  DateExpiry: 12:00
EndRequestForBids
```

Contacting bidders In general we can assume that all the agents in the network are accessible by any other, but obviously things are never this easy. We can use algorithm for parallel search like the ones used for acquaintance net in the allocation by delegation part, so when the process is completed the manager can choose the best bid. The problem is that the agents does not know their neighbors' skills.
A solution may be a **token ring** where the manager hands the announce through a ring passing by every other agents.



This is not fast since the execution is sequential.

Limit date If the manager waits for the previous process to complete we have the following problems:

- Agents who are not interested in the task have to responds anyway, increasing both the number of messages and the time spent waiting. The solution is to not send message if not interested.

- Agent that has experienced failure can block the system entirely. The solution is to add a deadline for receiving proposals.

Multiple manager and optimality There can be several manager issuing their request at the same time. Therefore we need to manage the concurrency for agents receiving multiple announcements since they can interfere with each other.
For Van Parunak the agents are ignorant in three ways:

- *Temporal ignorance*: since bidders are aware of the request already received and not those arriving [20].
- *Spatial ignorance*: Manager does not know about other requests for bids in progress while bidders cannot know about other proposal.
- *Ignorance of workloads*: bidders has no information of other agents' workloads.

Commitments, reservation and rejection of contracts Since we are working with concurrency we need to model the behavior in which new requests are receiven by bidders between the time they submit a proposal and the time they are rejected/awarded the contract. Should an agent which is waiting for the manager response take into account another proposal from another manager?

- If no, then the agent is working according to a **prudent** principle. Here the agent risk to end up without any work to do [21]. On the other hand the risk of overworking is minimized. This technique is useful when there *aren't too many agents* submitting proposals.
- If yes, then the agent is **brave** and it will not take into account the resource limit it has, and may end up with too much work to do. This method is more advantageous when there are a large number of agents in a competitive environment, where the probability of being awarded a contract is lower.
- There is a middle way in which we associate two values to a task: the probability of obtaining the contract and the estimated profit and chose to reject/accept the task based on those.

Influence of sub-contractors Another possible implementation is the one where a bidder (B) who has been awarded with a contract decides to brake it down into *sub-contract* [22]. The bidder than becomes a manager itself and request bids for the sub-contract. There are multiple possibilities:

[20] An agent can accept a task which is not optimal for him, when another one could have been accepted some time later.

[21] For example given the bidder A and two managers X, Y. A has just submitted its proposal to X (which will be rejected) and it's waiting for a response, in the meantime the announce from Y arrives (which will be accepted) and A reject it a priori. Then A ends up with no work to do.

[22] The same reasoning applies to this sub-contracts as the one for the sub-tasks. They must be as independent as possible to avoid redundant actions and cooperation.

- **Early commitment** B makes up a team of sub-workers to complete the assignment *after* submitting a proposal. The issue is when B cannot find any sub-workers for the sub-contract part, so it then tells its manager that he can no longer accept the contract and time is wasted.
- **Late Commitment** B makes up a team of sub-workers to complete the assignment *before* submitting a proposal. But we waste both time [23] and can be stuck in deadlocks.
- **Fixed agencies**: B is a boss of a small group of sub-workers which only accept assignments from B. This approach is less adaptive but more stable.

Pro/Cons of contract net Pros:

- Allocation mechanism is simpler to implement.
- Dynamic, no need for acquaintances management.
- Based on bilateral agreement which can depend on multiple parameters [24].

Cons:

- Many messages $O(nm)$, good only for small societies.
- Either the number of requests for bids is low or the data structures are complex for parallel implementation.
- When the task can be broken down into sub-tasks the agents need a decision-making strategy for choosing commitment mechanism.

Proposal guided contract net We can invert the contract net protocol so that suppliers alert potential clients to their capacity. So the systems becomes proposal guided by the availability of the bidders.

Contract net vs acquaintance net

- In acquaintances network agents' data [25] can be accessed directly by clients, while in contract net the agents have no priori knowledge of other agents, so they need to ask everyone.
- Acquaintance network lack dynamism, since any changes requires a network reorganization phase.

We can try to merge this two approaches in the following ways:

- Using acquaintance for small tasks which are immediate and contract for big ones.
- Acquaintance network can learn by using requests for bids every time it is necessary

Finally an acquaintance system can be seen as a cached memory of the contract net.

[23] If B is rejected then it must free its sub-workers
[24] Such as agents' skills, workload, type of task/data...
[25] Such as skills, state ...

4 Auctions

An auction can have the following proprieties:

- *Guaranteed success*: agreement is certain to be reached.
- *Maximizing social welfare*: maximizes the sum of the utilities of negotiation participants.
- *Pareto efficiency*: a negotiation outcome is said to be Pareto efficient if there is no other outcome that will make at least one agent better off without making at least one other agents worse off.
- *Individual rationality*: where playing by the rules is in the best interest of negotiation participants.
- *Stability*: all agents have an incentive to behave in a particular way, e.g. Nash equilibrium.
- *Simplicity*: appropriate strategy for negotiation is *obvious*.
- *Distribution*: there is no single point of failure and the communication between agents is minimized.

Mechanism Design

Structure An action takes place between an agent known as **auctioneer** and a collection of **bidders**. The goal of the action is to allocate the *good* to one of the bidders. While the auctioneer desires to maximize the price at which the good is allocated the bidders desires to minimize it.
There are many possible categories of auction based on:

- The *value* of the good:
  - **Common value**: when the good value is recognized the same by all the bidders, e.g. a dollar bill.
  - **Private value**: the good has different value for each agent, e.g. grandpa's socks.
  - **Correlate value**: the value depends both on common and private value, e.g. buy a something with the intention of selling it later.
- The *winner determination*, who gets the good?
  - **First-price**: the agent that bid the most is allocated the good.
  - **Second-price**: the agent that bid the highest is allocated the good, but pays the second-highest.
- The *secrecy of bids*:
  - **Open cry**: every agent know other agents' bids.
  - **Sealed bid**: agents are not able to determine bids made by other agents.

- The *auction procedure*:
  - **One shot**: single round of bidding.
  - **Ascending**: auctioneer starts from lowest price, bidders increase bids.
  - **Descending**: auctioneer starts from high value and start decreasing it, the first bidder to accept the bid is the winner.
- Number of auctioneers/bidders:
  - **Standard auction**: one seller, many buyers.
  - **Reverse auction**: many sellers, one buyer.
  - **Double auction**: many sellers, many buyers.
  - **Combinatorial auctions**: buyers and sellers may have combinatorial valuations for bundles of goods.

The winner curse When the auction is over, should the winner feel happy that they have obtained the good for less or equal to their private valuation? Or should they feel worried because no other agent valued the good as highly? The situation in which the winner overvalues the good is known as the *winner curse*.

4.1 Types of auctions

English Auction Is a *first-price, open-cry, ascending* type of auction.

- Protocol
  1. Auctioneer starts by offering the good at a low price.
  2. Bidders must offers higher prices.
  3. The good is allocated to the agent that made the highest offer.
- Proprieties
  - Generates competition between bidders (generates revenue for the seller when bidders are uncertain of their valuation).
  - Dominant strategy: Bid slightly more than current bit, withdraw if bid reaches personal valuation of good.
  - Winner's curse for common value goods.

Dutch auctions Is a *open-cry, descending* auction.

- Protocol
  1. Auctioneer starts by offering the good at artificially high value.
  2. Auctioneer lowers offer price until some agent makes a bid equal to the current offer price.
  3. The good is then allocated to the agent that made the offer.

- Proprieties
  - Items are sold rapidly (can sell many lots within a single day).
  - Intuitive strategy: wait for a little bit after your true valuation has been called and hope no one else gets in there before you (no general dominant strategy).
  - Winner's curse possible.

First-price Sealed-bid auctions Is a *one-shot, first-price, sealed-bid* auction.

- Protocol
  1. Within a single round bidders submit a sealed bid for the good.
  2. The good is allocated to the agent that made highest bid.
  3. Winner pays the price of highest bid.
- Proprieties
  - Problem: the difference between the highest and second highest bid is "wasted money" (the winner could have offered less).
  - Intuitive strategy: bid a little bit less than your true valuation (no general dominant strategy) [26].

Vickery auctions Is a *second-price, sealed-bid, one-shot* auction.

- Protocol
  1. Within a single round bidders submit a sealed bid for the good
  2. The good is allocated to agent that made highest bid.
  3. Winner pays the price of the second highest bid.
- Proprieties
  - Dominant strategy: bid your true valuation. If you bid more, you risk to pay too much. If you bid less, you lower your chances of winning while still having to pay the same price in case you win.
  - An antisocial behavior may occur in which a loosing agent [27] bid more than its true valuation to make opponents suffer (not "rational").

4.2 Lies and Collusion

There can be a collusion when groups of bidders cooperate in order to cheat. Bidders can agree beforehand to bid much lower than the public value so that when the good is obtained, the bidders can then obtain its true value and split the profits among themselves. All previous protocols are not collusion free, but they can prevent it by modifying the protocol so that bidders cannot identify each other.
There can be also be a lying auctioneer that places bogus bidders (shills) that artificially increase the price.

[26] The more bidders the smaller the deviation should be.

[27] The agent must know that it is going to loose to another agent and the bid the other agent will make.

4.3 Combinatorial Auctions

Until now we talked about *sequential* auctions where each good is sell sequentially and the determination of the winner is achieved by choosing the highest bid. A problem arises when the bidders have preferences over bundles [28]. In this case the bids will depend on *speculation* on what others will bid in the future.

We can either use parallel auctions or combinatorial auctions. In Combinatorial auctions the auctioneer puts several good on sale, each bidder must submit a bid for a bundle of goods. This is done in order to overcome the need for look-ahead [29] and the inefficiencies that arises from the related uncertainties.
In this kind of setting there is an auctioneer that has a set of items $M = \{1, 2, ..., m\}$ to sell, and there are N buyers which submit a set of package bid $\mathbf{B} = \{B_1, B_2, ..., B_n\}$, where $B_i = (S_i, p_i)$, where $S_i \subseteq M$ is a set of items and $p_i \geq 0$ is a price.
The **winner determination problem** is the problem of maximizing the sum of accepted bid prices:

$$max \sum_{j=1}^{n} p_j x_j$$

under the constrain that each item is allocated to at most one bid:

$$\sum_{j|i \in S_j} x_j \leq 1, \ \in \{1..m\}, \ x_j \in \{0, 1\}$$

and each agent receives at most one bundle:

$$\sum_{S \subseteq M} x_{S,i} \leq 1, \ \forall i \in N$$

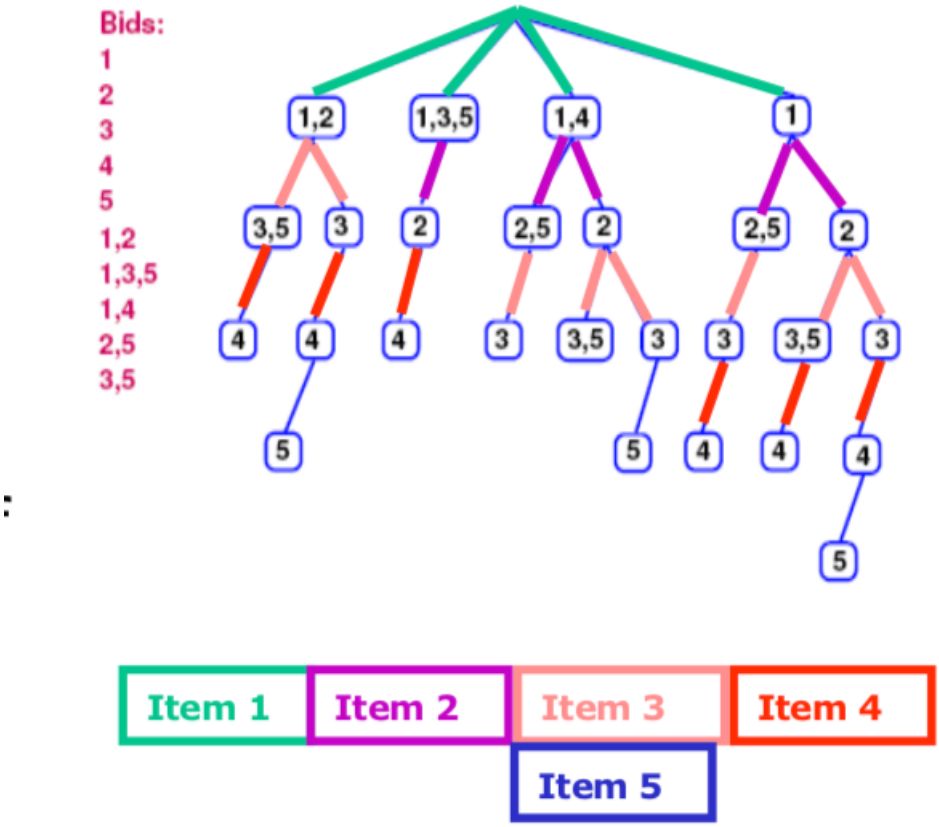
This problem is NP-complete and inapproximable.
We can use heuristic search to solve the problem representing the states with two possibilities.

Branch-on-items The question is *What bid should this item be assigned to?* So the state is represented by a set of items which have already been allocated and the branching is carried out by adding further items.
Each path on the search tree is a sequence of bids that do not share the same items with each other [30] and it ends when there is no other bids to add.

[28] A bundle is a set of goods.

[29] In which an agent will try to base its bid on the behavior of other. The results is that everyone will wait for someone to make the first move and a deadlock will occur.

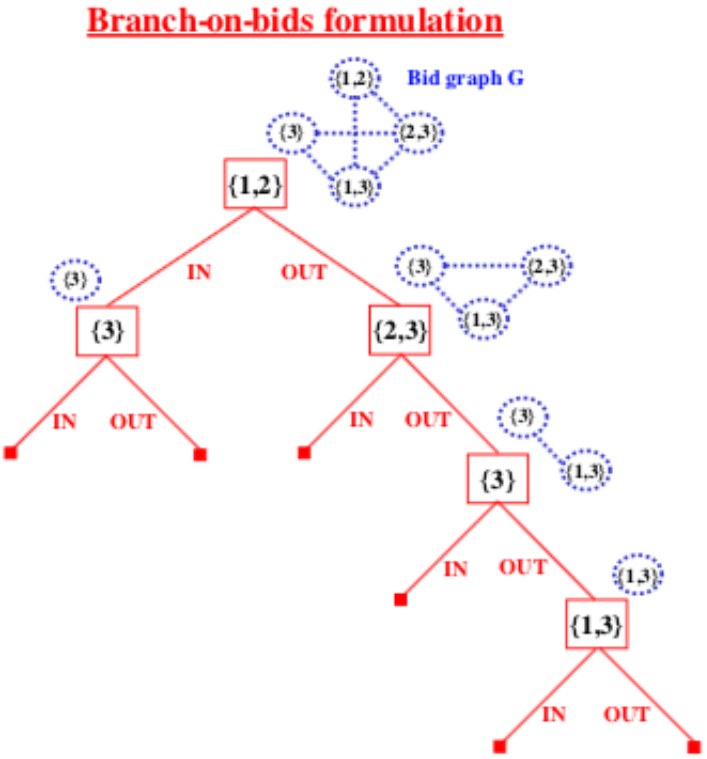
[30] Disjoint bids.



As the search proceeds down a path a tally g is kept of the sum of the prices; at every node the revenue g form the current path is compared with the best $g - value$ found so far to determine if the current path is the best solution.
But the *optimality* of the search my fall when there are some items with no bids or when the auctioneer makes better profit when keeping a bit. For example there is no bid for 1, there is a 5$ bid for 2 and a 3$ bid for [1,2]; the auctioneer makes better profit by selling *only* 2. So he can make use of **dummy bids** of prices zero on those items with no 1-item bid.
Moreover we can reduce the brancing factor of a tree by remembering that the order of bids on a path does not matter. Finally the number of leaves can be no grater than $(\frac{n+n_{dummy}}{m})^m$, hence in the worst case the size of the tree is polynomial in the number of bids, but exponential in the number of items.

Branch-on-bids In this kind of search the question is *should this bid be accepted or rejected?* [31]. The children in the search tree are the world where that bid is accepted (IN) and the world where the children is rejected (OUT).

[31]Since the possible answer is either yes or no, the tree is a *binary tree*, so the branching factor is 2.



As you can see for the first bid (1,2), the children are the worlds in which it is accepted (left, IN) and rejected (right, OUT).
For this kind of search **no dummy bids** are needed, and the number of leaves is *at most* 2^n. Moreover this search is in line with the principle of *least commitment* [32], therefore we can refine the search control by ordering bids.

Heuristics Given $f(n) = g(n) + h(n)$ we have that:

- *g(n)* is the revenue generated by bids that were accepted along the path.
- *h(n)* estimates for every remaining node in the current state, how much additional revenue can be expected.

Moreover we can prune large parts of state space by using **upper bounds** on how much the unallocated items can contribute; we simply sum over the unallocated items of the item's maximum contribution:

$$\sum_{i \in A} c(i), \ \textit{where } c(i) = \max_{j | i \in S_j} \frac{p_j}{|S_j|}$$

Tighter bounds are obtained by recomputing $c(i)$ every time a bid is appended to the path.
Finally we can use the linear program relaxation of the remaining winner to determine the problem upper bound.

[32] Partial order planning.

4.4 Parallel Auctions

Multiple items are open for auction simultaneously, bidders may place their bids during a certain time period, and the bids are publicly observable. Uncertainty and need for look-ahead are *lowered* since their are partial signal to the bidder on what the others' bids will end up being.
Since each bidder would like to wait until the end to see what the going prices will be [33], there is a chance that no bidding would commence. As a patch to this problem, activity rules have been used.

[33] In order to optimize its bids so as to maximize payoff given the final prices.

5 Distributed Constrain Optimization Problem [DCOP]

In DCOPs for MAS agents each agent negotiates locally with just a subset of other agents, called neighbors. We work in a decentralized fashion for the usual reasons, such as *robustness* [34] and *scalability* [35]. On the other hand this approach can have some complications such as the need for *distributed knowledge* (since there is no centralize structure) *information sharing* (i.e. the need of coordination) and *complex systems* (which are difficult to design).
In this kind of problems each agent is considered *benevolent*, that is the agents prefer to lose some optimization on its own valuation function in order to maximize the global function.

Constrain Networks A constrain network $\mathcal{N}$ is defined as a tuple $\langle X, D, C\rangle$ where:

- $X = \{x_1, x_2, .., x_n\}$ is a set of *discrete* variables
- $D = \{D_1, D_2, ..., D_n\}$ is a set of variables domains for the corresponding variables, e.g. $x_i \in D_i$
- $C = \{C_1, C_2, ..., C_m\}$ is a set of constrains which can be:
  - **Hard**: when it defines a relationship R_i between a subset of variables $S_i \subseteq X$, so that R_i enumerates all the *valid* assignment of S_i. This defines a *Constraints Satisfaction Problem* CSP [36] in which our aim is to *find an assignment* for all the variables in the network that satisfy *all the constraints*.
  - **Soft**: when there is a function F_i which maps every possible assignment of the subset S_i to a real value. In this case we're facing a *Constraints Optimization Problem* COP in which our aim is to find the *best* solution rather than a valid one. So we need to define a global function $F(\overline{a})$ which is an aggregation of all the local functions $F_i(\overline{a}_i)$ in the problem, in which $\overline{a}_i$ are the restriction of $\overline{a}$ to S_i.

It is interesting to notice how a CSP can be turned into a COP by assigning a fixed cost to every violated constraint and search for an assignment that minimizes the sum of costs, this problem is known as **Max-CSP** problem.

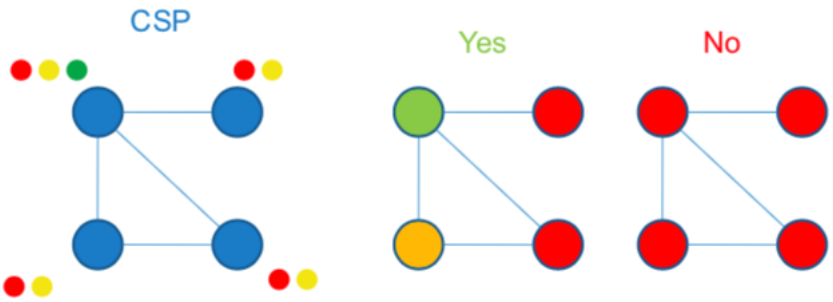
Distributed Constraints Processing A DCOP can be represented as a constraint network $\mathcal{N} = \langle X, D, C\rangle$ plus a set of k agents $A = \{A_1, A_2, ..., A_k\}$, where each agent:

- Can control *just one* variable, for simplicity.
- Is aware of the constraint that involves *only* the variable that it controls.

[34] Avoid single point of failure.
[35] Avoid excessive bandwidth allocation.
[36] Studied in the search part.

- Is considered a neighbor of another agents if there is *at least one* constraint that depends on both. For this reason only neighbor agents can communicate with each other, which minimizes the amount of information that they must reveal to each other, thus preserving their privacy.

Benchmarking problems For example the *graph coloring* problem which is a simple to formulate problem that can be easily modeled by a set of parameters [37]. In this graph nodes are variables, the set of k colors is the variable domain and the constraints are *not-equal* that old between adjacent nodes. For example the following Figure is an instance of CSP on the graph coloring problem:



Complete Algorithms We now focus on complete solutions techniques [38], which are NP-Hard so the exhibit an exponentially increase in coordination overhead [39] as the number of agents increases.
They can be divided in the following categories:

- *Search based* algorithm, which can be divided into
  - *Synchronous* such as SyncBB, AND/OR search. Here the agents waits for messages from other agents before computing and sending messages themselves, which minimizes redundancy and uses only essential information.
  - *Asynchronous* such as ADOPT, NCBB and AB. Here agents performs computation and sends out messages without waiting, which makes use of parallel computation.
  - *optAOP* which is an algorithm that discovers pieces of the problem that are particularly hard to solve in a decentralized fashion, and centralizes them into sub problems that are delegated to mediators.
- *Heuristic driven* algorithm such as DSA,MGM, MAX-sum, which are *not* guaranteed to find an optimal solution but have minimum coordination overhead agents.
- *Approximate approaches* such as Bounded max-sum, k-optimality, that both guarantees optimality and keeps a low coordination overhead.

[37] Like number of available colors, ratio between number of constraint and number of nodes.
[38] Techniques which always find a solution.
[39] Either size/number of messages or individual agent computation.

5.1 Dynamic Programming Optimization Protocol DPOP

This algorithm is based on the distributed version of the *Bucket Elimination* algorithm. It guarantees that the optimal solution can be found with a linear number of messages, and it has three phases:

1. Arrangement of variables into a DFS [40] tree [41].
2. Propagation of *Util* messages bottom-up along the DFS tree. In this part information are compiled to compute optimal values.
3. Propagation of Value message top-down the DFS tree, given the previous information the root chooses an optimal value and propagate the decision.

5.1.1 Pseudo-tree Ordering

Given a graph G, a pseudo-tree is a rooted tree in which adjacent nodes[42] of G fall under the same branch [43], i.e. belongs to the same path from the root to a leaf. It is easy to understand how a tree like this can be build using *Depth First Search* (we will call the result DPS-tree and it will be a sub-class of the pseudo-tree) , since a path in it will be composed by only adjacent nodes. To build a pseudo-tree we first need an **ordering** of the nodes [44] and the algorithm is the following:

1. Traverse the graph using a recursive procedure starting from the given *ordering*.
2. Each time we reach a node x_i form x_j we mark x_i as visited and we say that x_j is the father of x_i, effectively creating a **tree edge**.
3. When x_i has a visited neighbor x_k that is not in the parent path we state that x_k is a pseudo-parent of x_i, effectively creating a **back edge**.

Example Given the graph $\mathcal{G}$ (Figure 1) where:

$$\mathcal{G} = \begin{cases} C = \{X, D, C_{hard}, C_{soft}\} \\ X = \{a, b, c, d, f, g\} \\ C_h = \{\} \\ C_s = \{F_b^a, F_d^b, F_b^a, F_c^a, F_c^b, F_f^b, F_f^c, F_g^f\} \end{cases}$$

[40] Depth First Search.
[41] Can be build by token passing algorithm.
[42] Variables which share some kind of constraint.
[43] This leads to the conclusion that once a node is instantiated, its sub-trees are completely independent.
[44] Different ordering will lead to different pseudo-tree. Moreover finding the optimal ordering can be reduce to find the ordering which minimizes the separator Sep_i size.

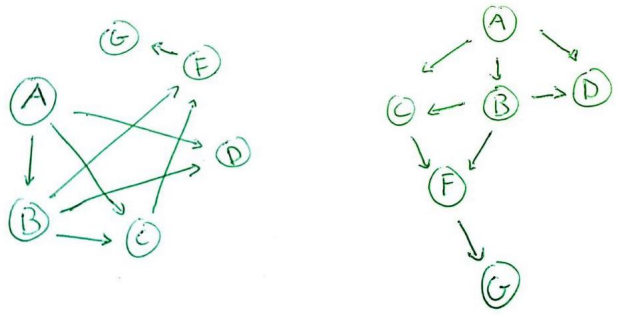


Figure 1: Graph $\mathcal{G}$

And given the order $o : \{a, b, d, c, f, g\}$, the resulting DFS-tree $\mathcal{P}$ is:

- The Levels are: $L_0 = \{a\}, L_1 = \{b\}, L_2 = \{c, d\}, L_3 = \{f\}, L_4 = \{g\}$
- The Tree edges are: $\{F_b^a, F_d^b, F_c^b, F_f^c, F_g^f\}$
- The Back edges are : $\{F_d^a, F_c^a, F_f^b\}$

As shown in Figure 2

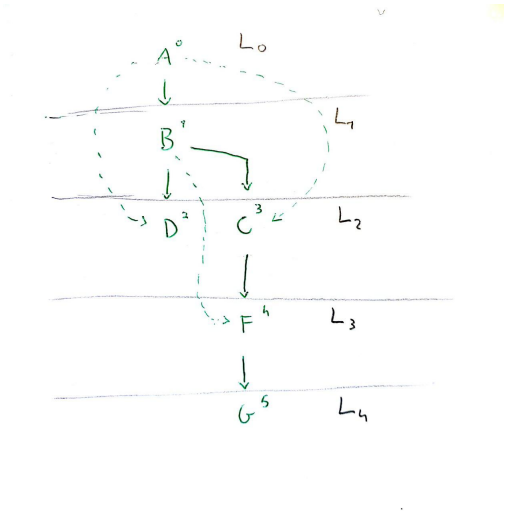


Figure 2: DFS-tree $\mathcal{P}$

Concepts Given two nodes x_i, x_j we have that:

- If x_j is an ancestor of x_i through a *tree edge*, then x_i is a **children** C of x_j, while the latter is the **parent** of x_i.
- If x_j is an ancestor of x_i through a *back edge*, then x_i is a **pseudo-children** C of x_j, while the latter is the **pseudo-parent** of x_i.

- A **separator** of x_i, Sep_i is a split (so a set of nodes) of the tree in two parts: one part is made up of all the ancestors which are connected with x_i [45], the other one is made up of all the descendants of x_i. This set is called separator because it is precisely the set of agents that should be removed to completely separate the sub-tree rooted at x_i from the rest of the network.
- A **minimal separator** of x_i are all the nodes that, if removed, completely disconnects the sub-tree rooted at x_i from the rest of the problem.

Building DFS-tree in distributed system We can use the Token passing algorithm shown in Figure 3.

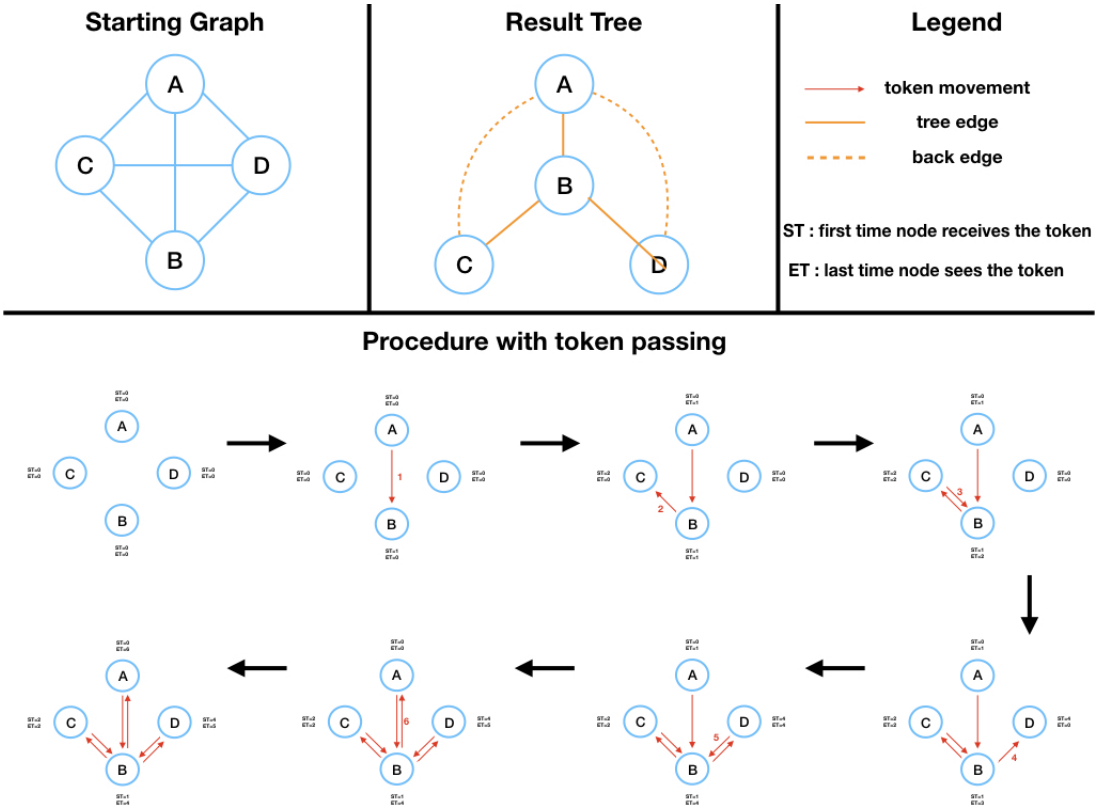


Figure 3: Token passing algorithm

Good heuristics Finally some good heuristic for this kind of problem are: Maximum Connected Node MCM and Maximum Cardinality Set for DFS.

5.1.2 Util Propagation

As before mentioned the Util propagation phase has a *bottom-up approach* in which each node, starting from the leaves, computes a message *for* its parent

[45] Either through tree or back edges.

considering both: the messages received from its children and the constraints that agent is involved in.
The goal is to build a value function so that the root agent can make an optimal decision.
The formula for the Util message $U_{i\to j}$ that agent A_i has to sent to its parent A_j is:

$$U_{i\to j}(Sep_i) = \max_{x_i}\left(\bigoplus_{A_k\in C_i} U_{k\to i} \oplus \bigoplus_{A_p\in P_i\cup PP_i} F_{i,p}\right)$$

s Where:

1. C_i is the set of children of agent A_i.
2. P_i is the set of parent A_i.
3. PP_i is the set of agents preceding A_i that are connected thought a *back edge*, i.e. its pseudo-parents.
4. Set_i are the set of agents preceding A_i that are connected either with A_i or with one of its descendant.
5. $A_p \in P_i \cup PP_i$ are all the agents A_p which are either parents or pseudo-parents of agent A_i.
6. The $\oplus$ operator is a join operator that sums up functions with different but overlapping scopes consistently, i.e. summing the values of the functions for assignments that agree on the shared variables.
7. $\bigoplus_{A_p\in P_i\cup PP_i} F_{i,p}$ is the joint summatory of the values driven by the soft constraints, either back or tree edges, for every parent, pseudo or non, of A_i.
8. While $A_k \in C_i$ are all the children of A_i
9. And $\bigoplus_{A_k\in C_i} U_{k\to i}$ is the joint summatory of all the messages coming from the children of A_i.
10. So that $\left(\bigoplus_{A_k\in C_i} U_{k\to i} \oplus \bigoplus_{A_p\in P_i\cup PP_i} F_{i,p}\right)$ is the joint sum of both the constraints carried from the children and the constraints A_i has towards its parents.
11. Finally the max operation is due to the necessity to take the value which maximizes the entire function.

Example Given the DFS-tree in Figure 4

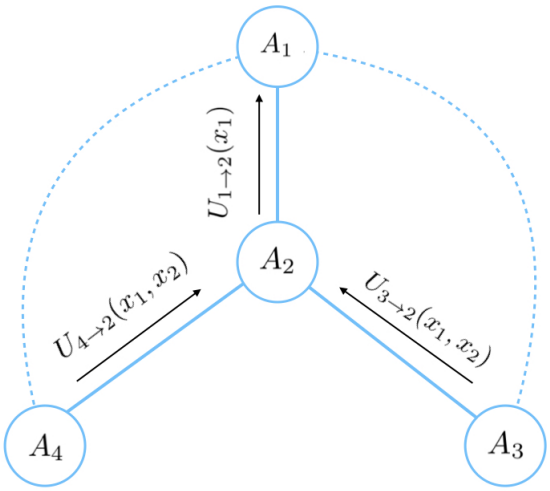


Figure 4: Util propagation

We have that the values of the Util massages are the following:

- $U_{3\to 2}(x_1, x_2) = \max_{x_3}(F_{3,1}(x_3, x_1) \oplus F_{3,2}(x_3, x_2))$, where:
  - there is no contribution from 9 since A_3 has no children.
  - $P_i \cup PP_i = \{A_2\} \cup \{A_1\} = \{A_2, A_1\}$ so that
  - $F_{3,1}(x_3, x_1) \oplus F_{3,2}(x_3, x_2)$ is driven by 7.
- $U_{4\to 2}(x_1, x_2) = \max_{x_4}(F_{4,1}(x_4, x_1) \oplus F_{4,2}(x_4, x_2))$, same as above
- $\max_{x_2}(U_{3\to 2}(x_1, x_2) \oplus U_{4\to 2}(x_1, x_2) \oplus F_{2,1}(x_2, x_1))$, where:
  - $C_2 = \{A_3, A_2\}$ so that
  - $U_{3\to 2}(x_1, x_2) \oplus U_{4\to 2}(x_1, x_2)$ is generated by 9,
  - plus the part generated by 7

Message composition Given a graph coloring example, whit the graph in Figure 4, in which we can have either black or white (0,1). We first consider the agent A_3 and we compute the Table 1. In this table we assign -1 if the is not respected (same color), 0 otherwise.

x_1	x_2	x_3	$F_{3,1}$	$F_{3,2}$	$F_{3,1}(x_3, x_1) \oplus F_{3,2}(x_3, x_2)$
0	0	0	-1	-1	-2
0	0	1	0	0	0
0	1	0	-1	0	-1
0	1	1	0	-1	-1
1	0	0	0	-1	-1
1	0	1	-1	0	-1
1	1	0	0	0	0
1	1	1	-1	-1	-2

Table 1: $U_{3\to 2}(x_1, x_2)$

5.1.3 Value Propagation

In this last phase the root agent A_r computes X_r^* [46] which is the argument that maximizes the sum of the messages received by all its children.
The generic agent A_i computes:

$$x_i^* = argmax_{x_i}(\sum_{A_j \in C_i} U_{j\to i}(x_p^*) + \sum_{A_j \in P_i \cup PP_i} F_{i,j}(x_i, x_j^*))$$

The computation is very similar to the one in the Util propagation, in which we have two pieces of the equation. One representing the optimal values chosen for the current agent's parents, while the other one is the function specific of the agent. In particular:

- $x_p^* = \cup_{A_j \in P_i \cup PP_i}\{x_j^*\}$ is the set of optimal values of A_i's pseudo/parents.
- So that $\sum_{A_j \in C_i} U_{j\to i}(x_p^*)$ is the sum of all maximized values coming for the parent choices.
- On the other hand $F_{i,j}(x_i, x_j^*)$ is the value of the agent A_i function given the optimal values coming from its parent, and its own constraint.

After this computation, agent A_i sends to its children A_j the message:

$$V_{i\to j} = \{x_i = x_i^*\} \cup \bigcup_{X_k}\{x_s = x_s^*\}, \quad X_k = X_s \in Sep_i \cap Sep_j$$

in which:

- $\{x_i = x_i^*\}$ is the optimal value A_i has chosen for itself.
- X_k are those agents who are pseudo/parents of both A_i, A_j so that
- $\bigcup_{X_k}\{x_s = x_s^*\}$ are tho optimal values this pseudo/parents have chosen for themselves.

Since the maximization formula is the same as the one in the Util propagation phase, we can store the values in a table, during the Util propagation phase, hence reducing the computation.

[46] So actually deicing the value of its own node and leaving the rest to decide.

6 Swarm Robotics

Swarm robotics studies robotic systems composed of a multitude of interacting units. We can either gave homogeneous systems or few heterogeneous groups in which each unit is relatively simple and inexpensive. Single units have limitations, either physical of functional, and may lack global information.

6.1 Random Walks

We need a basic search strategy that can be driven from the animal word based on the following *missing assumption*:

- No individual knowledge
- No memory of the path taken
- No learning abilities

Although there must be some kind of information sharing between agents that leads to a trade-off between ability ti widely search the environment and ability to share information [47].
A random walk can be broken down to a sequence of alternate **straight motion** and **random turns**. There are two approaches for this kind of motion:

6.1.1 Correlated Random Walks (CRW)

In this kind of walks there is a positive correlation between consecutive movements, that is, walkers moves in similar directions with high probability.

Turning angle The turning angle θ is chose from a wrapped Cauchy distribution with the following PDF [48]:

$$f_w(\theta; p) = \frac{1}{2\pi} \frac{1 - p^2}{1 + p^2 - 2pcos(\theta)}, \quad 0 < p < 1$$

in which the parameter p determine the skewness of the distribution as shown in Figure 5, the higher p the less likely is the robot to deviate from its current direction.

The correlation between movements is given by the $1 + p^2 - 2pcos(\theta)$ parameter that, for higher thetas, decreases the output so that the orientation never changes too much. For $p = 1$ we obtain a *Dirac* distribution corresponding to a straight line motion, while for $p = 0$ the distribution becomes uniform and provides no correlation between consecutive movements.

Straight motion For straight motion step, we define a distribution with finite second moment, so that the latter tends to a normal distribution according to central limit theorem.

[47] This kind of problem may be tackled with special reverse behavior such as homing to a central place.
[48] Probability density function.

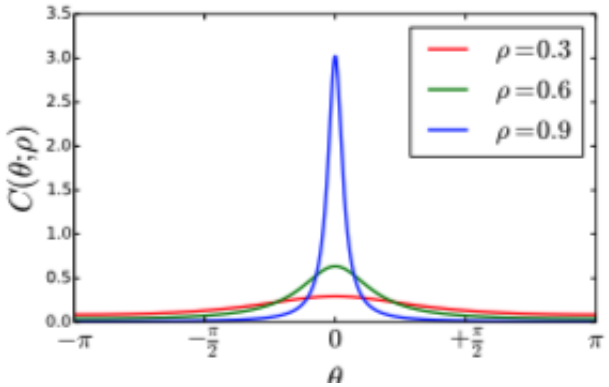


Figure 5: PDF for CRW with different values of p

6.1.2 Lévy Walks (LW)

This kind of random walk is characterize by several small displacement interleaved by long relocations. This provide an optimal search behavior for sparse targets since the long motion steps allow to search in different areas without frequently passing over sites that have been already visited.
The LW is characterized by a step-length distribution that follows a power law:

$$P_\alpha(t) \sim t^{-(\alpha+1)} \quad 0 < \alpha \leq 2$$

which can be defined in terms of Fourier transformation:

$$F(k) = e^{\beta|k|^\alpha}$$

for $\alpha = 2$ the distribution becomes a Gaussian, while for $\alpha \to 0$ the random walk reduces to straight line paths.

6.2 Aggregation

Aggregation is a collective behavior that leads a group of agents to gather in some place. Therefore there is a transition from a homogeneous to a heterogeneous distribution of agents.
Depending on the presence of heterogeneities [49] in the environment we can have an easy problem. On the other hand, when no form of pre-existing heterogeneities can be found, the problem becomes more complex and an aggregation algorithm must be used.

There are two kind of aggregation algorithms:

- **Positive feedback** : in which agents emits signals that diffuses in space. The signal of neighboring agents sum up and becomes more attractive leading to the formation of large aggregates (Figure 6).

[49] Heterogeneities (such as corners, shelters, humidity ecc...) can be exploited for aggregation.

- **Social cues** : agents move randomly with certain probability to remain still for some time. The vicinity to other agents increases the probability of stopping and of remaining within the aggregate, eventually producing an aggregation process mediated by social influences (Figure 7). Aggregation is dependent on two main probabilities: the probability to enter an aggregate, which increases with the aggregate size, and the probability to leave an aggregate, which decreases accordingly.

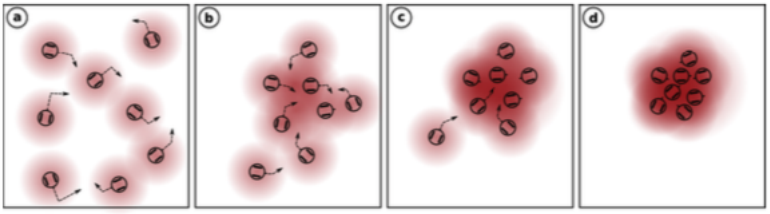


Figure 6: Positive feedback approach for swarm aggregation

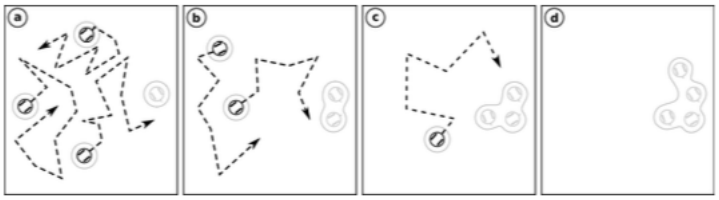


Figure 7: Positive feedback approach for swarm aggregation

6.3 Coordination

In swarm robotics, ensuring coherence in space [50] is a fundamental propriety. We can have coordinated motion based on self-organization since we may have a non-uniform distribution of information, i.e. having some agents that are more informed than the others on a preferred direction of motion.
We can have agents that are influenced solely by its nearest neighbor. Also, the movement of each agents is based on the same behavioral model, which includes also some inherent random fluctuation. We can use just three rules to model a fine behavior:

- Approach faraway individual.
- Get away from agents that are too close.
- Align with the neighbor direction.

As shown in Figure 8, when the nearest neighbor is within the closest region, the agent reacts by moving away. When the nearest neighbor is in the farthest region, the agent reacts by approaching. Otherwise, if the neighbor is within the intermediate region, the agent reacts by aligning.

[50] That is display coordinated movement in order to maintain a consistent spatial structure.

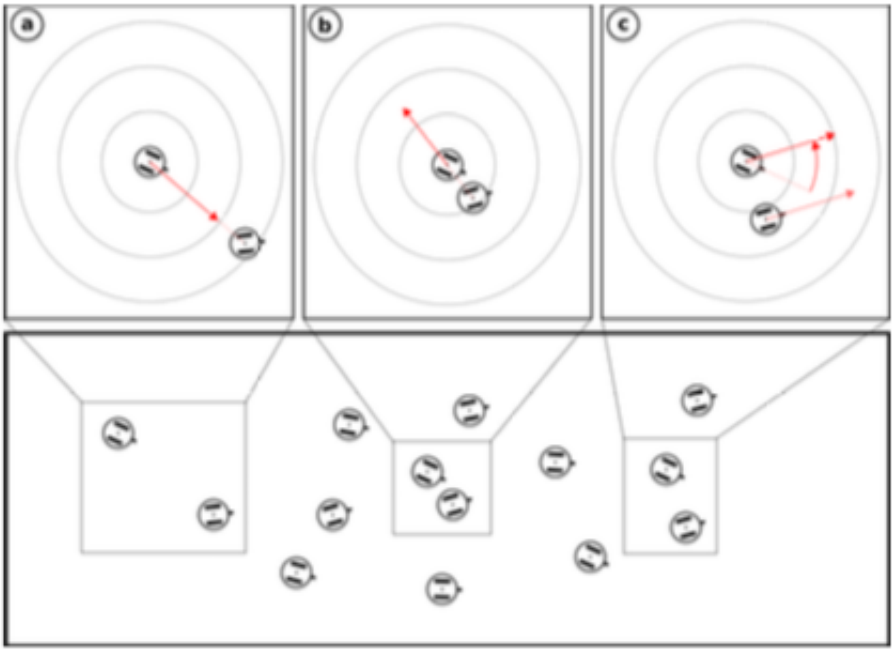


Figure 8: Coordination of swarm robots

6.4 Collective Exploration

Usually the swarm cannot completely perceive the environment, and the environment may also change during the operation of the robots; hence we need to lean on some common characteristic found in swarm robotic scenarios:

- A **central place** is a specific location where robots must come back regularly. A scenario that involves a central place requires a swarm able to either remember or keep track of that location.
- If the environment is closed (**finite area**) and not too large, the swarm may use random motion to explore, with fair chances to rapidly locate resources (or even the central place). In an open environment, robots can get lost very quickly. In this type of environment, it is necessary to use behaviors that allow robots to stay together and maintain connectivity.
- **Obstacles** are environmental elements that constrain the motion of the swarm. If the configuration of the obstacles is known in advance, the swarm can move in the environment following appropriate patterns. In most cases, however, obstacles are unexpected or might be dynamic and may prevent the swarm to explore parts of the environment.

CE in swarm robotics Animals also heavily rely on random motion to explore their environment . Usually the exploratory pattern is not fully random (that is, isotropic), because animals use all possible environmental cues at hand to guide themselves. Random motion can be biased towards a given direction,

or it can be constrained in a specific area, for instance around a previously memorized location. For this reason the robot must have some kind of memory to avoid redundant search, either by means of localization techniques or by mapping the environment.
A collective motion aims at maintaining the cohesion between agents while moving through the environment. There are two possible implementations:

- The swarm can behave like a sort of physical mesh that covers part of the environment. This is called the **gas expansion** behavior where one robot keeps track of the central place while other agents spread as far as they can from each other while maintaining visual contact.
- On the other hand swarms may form a **chain** whit one end that sticks to a central place and the other that moves freely through the environment [51].

6.5 Task allocation

A *response threshold model* assumes that each worker responds to a given stimulus when the stimulus intensity exceeds the worker's threshold. By allowing the threshold to vary in time with a reinforcement process [52] we can have a more robust, general end long-time scaled model

Static model Consider:

- m tasks that need to be performed
- $\mathcal{N}$ workers, denoted by i with
- $\theta_{ij} (i = 1, ..., \mathcal{N}; j = 1, ..., m$ response threshold, associated with task j.
- s_j is the intensity of task j.
- $\mathcal{T}_{\theta_{ij}}(s_j)$ the probability for worker i and task j, of "accepting the task".

In the fixed threshold model we have the probability of response equal to:

$$\mathcal{T}_{\theta_{ij}}(s_j) = \frac{s_j^2}{s_j^2 + \theta_{ij}^2}$$

We have that:

$$\begin{cases} \mathcal{T}_{\theta_{ij}}(s_j) \to 0, & \text{if } s_j \ll \theta_{ij} \\ \mathcal{T}_{\theta_{ij}}(s_j) \to 1, & \text{if } s_j \gg \theta_{ij} \\ \mathcal{T}_{\theta_{ij}}(s_j) = 0.5 & \text{if } s_j = \theta_{ij} \end{cases}$$

That is lower θ_{ij} are more likely to respond to lower level of stimulus.

[51] Allow to cover more area than the gas expansion behavior.
[52] A threshold decreases when the corresponding task is performed, and increases when it is not performed.

Dynamic model On the other hand we can have a time dependent threshold, given:

- ζ: the coefficient that describes the learning
- ϕ: the coefficient that describes the forgetting

both assumed to be identical for all tasks.
An individual i becomes *more* sensitive [53] by an amount $\zeta\delta t$ to a stimulus s_j associated with the task j when performing that task during the time period of duration δt:

$$\theta_{ij} = \theta_{ij} - \zeta t$$

On the other hand, an individual i becomes *less* sensitive by an amount $\phi\delta t$ to a stimulus s_j associated with the task j when *not* performing that task for a time period of duration δt:

$$\theta_{ij} = \theta_{ij} + \phi t$$

Finally, given x_{ij}, i.e. the fraction of time spent by agent i in performing taks j: the agent performs the task during $x_{ij}\delta t$ whitihn δt, and other tasks during $(1 - x_{ij})\delta t$. The resulting cchange in θ_{ij} within δt is given by:

$$\theta_{ij} = \theta_{ij} - x_{ij}\zeta\delta t + (1 - x_{ij})\phi\delta t \quad \theta_{ij} \in [\theta_{min}, \theta_{max}]$$

6.6 Collective decision making

With swarm robotics it is frequent that agents have different opinion about the correct decision [54], hence a mean collective decision is needed.

Variants There are three kind of variants for decision making in swarm robotics:

- **Opinion propagation**: As soon as a group member has enough information about a situation to make up its mind, it propagates its opinion through the whole group. It allows for fast collective decision at the cost of robustness and accuracy, since messages can be corrupted.
- **Opinion averaging**: All individuals constantly share their opinion with their neighbors and also adjust their own in consequence. This leads to a collective decision by the mean of an averaging function. It is more robust than the previous one but it works best when all individuals have roughly identical knowledge [55]
- **Opinion amplification**: All individuals start with an opinion, and may decide to change their opinion for another one. The switch to a new opinion happens with a probability calculated on the basis of the frequency of this opinion in the swarm. Practically, this means that if an opinion is more represented in the group, it has also more chances to be adopted by an individual, which is why the term amplification is used. Instead of adjusting opinions, agents adopt new opinion with some probability, which prevents the lack of good knowledge to result in poor decision making.

[53] I.e. the agent has a better knowledge of task j, thus in the future it will be more likely to choose task j for its familiarity.
[54] Based on partial/incorrect information.
[55] When there are more individuals that have poor information, these are the one which will be chosen more often (oddly familiar).

Decision making in swarm robotics Here, robots can share and merge their localization opinions when they meet, by means of local infrared communication. By doing so, robots manage better localization and improve their performance in the foraging task. Moreover, robots associate a confidence level to their estimates, which is used to decide how information is merged. If a robot advertises an opinion with a very high confidence, then the mechanism produces opinion propagation. Hence the two mechanisms of averaging and propagation are blended in a single behavior, and the balance between them is tuned by the user with a control parameter.