## **Multiagent systems** :

- Centralized multiagent system : The agents are runned under the same platform.
  Can be :
  - Egalitarian : minimum reward for each agent (little room for optimization).
  - Social welfare : optimize the sum of rewards and redistribution.

  Two-step procedure :
  - Offline : Compute the value function for every agent using Markov decision process.
  - Online : Use the combination of agents and actions that can optimize the reward.

  Many disadvantages compared to distributed multiagent system (agent run on different platforms and coordination through message exchange) like if the central agent fail, the whole planning fail.

- Joint value function : Linear combination of local value function.

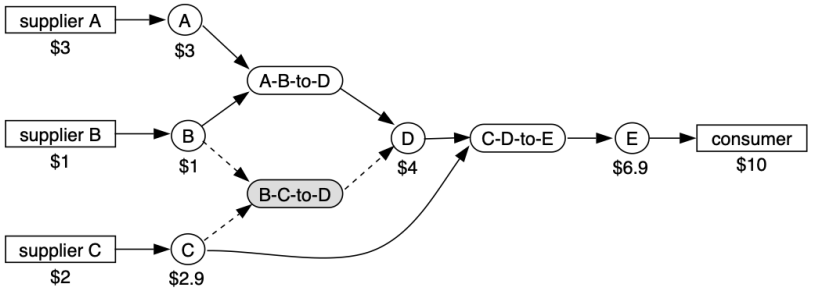
- Partial-global-planning : Used in distributed multiagent system. Example of a blackboard system : meeting room where the mediator can find conflicts and synergies when each agent present their goal/plans. The partial plans (plan for one agent) are combined to make a global plan. If there is a scheduling problem can resolve this as a constrained optimization problem.

- Publish-subscribe systems : An agent publish and the others agents are notified to share joint goals.

- Ontologies : Communication between agents the basic as possible but try to remove disambiguation. Knowledge based network where ontologies are classed into classes and subclasses. Add common-based knowledge to restrict properties of ontologies. Like the semantic web. Equivalent to the semantic network. (OWL : Web Ontology Language)

- Contract Nets : Make auctions so that the agent that has better utility will make the task. The agents can change between auctioneers and contractors. However, conflicts may happen as the first come, first served. FIPA is the language used for an auction.

- Market-based contract nets : Market consisting of suppliers, producers and consumers. The suppliers and consumers perform a multi-unit auction (e.g. (M+1)st auction or McAfee auction). The bids can update through message exchanges. In order to make a feasible deal, an incrementing bid for the producer is done until the deal is feasible. In that case, C increase until A-B-to-D is prefered over B-C-to-D so that C can be used for C-D-to-E and the deal will be feasible. Don't work for asynchronous settings.



## **Distributed Multiagent Systems** :

- Social laws : Laws that every agent follow, no need of messages exchanged (e.g. drive on the right of the road). Find good social laws is NP-complete in number of states.

- Secondary market : Calculate the marginal cost for an agent and a task

  $$c_{add}(A_i, t) = cost(A_i, T \cup t) - cost(A_i, T)$$

  . An announcer will pay for less than its marginal cost while the bidder will want to be paid higher than its marginal cost. The agents can exchange the tasks between themselves. Ma,y drawbacks like the time it take until knowing a winner, management of message exchanged, the convergence is either very long or impossible.

- Constraint satisfaction problem : How the tasks and resources should be allocated can be expressed as a constraint satisfaction problem.

  Given $< X, D, C, R >$:
  - variables $X = x_1, ..., x_n$
  - domains $D = d_1, ..., d_n$
  - constraints $C = c_1(x_{i,1}, x_{k,1}), ..., c_m(x_{i,m}, x_{k,m})$
  - relations $R = (r_1 = \{(v_1, v_2), (v_3, v_4), ...\}, ..., r_m = \{(v_o, v_p), (v_q, v_r), ...\})$,

  Multiple algorithms to find a solution to the distributed constraint satisfaction problem.
  - Synchronous backtracking : Create a tree search for all solutions. Give a task for each agent and if the solution don't satisfy the constraints, then backtrack. Can use a CSP heuristic function. Two methods :
    - Forward checking : Keep track and assign task until the domains are empty.
    - Dynamical variables ordering : Take the agent that have the smallest domain.
  - Asynchronous backtracking : Agents can work in parallel and attribute themselves a task. They send a message to other agents to see if the constraints are satisfied. If no satisfaction, go to the task with lower priority and no loop is created.

- Distributed upper confidence tree : A constraint graph is a graph containing each variables as a node and the variables that share a constraint in common have a link between them. A pseudotree use the constraint graph by saying that variables sharing a constraint must communicate with each other but not all edges are necessary for communication (back-edges). The pseudotree is a rooted tree, every node has exactly one parent node. Solve the pseudotree with Monte Carlo tree search and upper confidence bound for sampling the evaluating node.

- Dynamic programming : If two variables share a constraint, their utility is expressed in terms of both variables. Find a Nash equilibrium solution through message exchanged $utility(x=v) = utility(x=v,y=w)$. In term of rooted tree, for each agent, each variable send the combined constraints to their parent nodes which sum up their utilities and inform the agent of the action it takes. To reduce the number of messages while increasing the message size, one can use a pseudotree where one variable send the combined constraint of the parent and back-edges nodes.

- Local search : Initialize the variables to arbitrary values, choose the best neighbors to improve the solution until no improvements are found. It is simple to implement but often it is found stuck in a local minima. Further improvements can be made :
    - Distributed min-conflicts : Assign random values in parallel, search changes that reduce the number of conflicts. Two neighbors (variables sharing a constraints) can't change in the same step.

    - Asynchronous min-conflicts : If one variable change value, it send ok? message to neighbors. The neighbors will then send ok message back if it is ok.

    - Breakout algorithm : Assign weights to every constraints. Modify the variables that reduce the most the sum of weight for the constraints. When being stuck in a local minimum, the weights of the existing constraints are increased. It will create a breakout where new conflicts will be more profitable than existing ones. Another tricky part is to know when all the conflict have been resolved. The agent exchange message, they send 0 when a conflict is present and +1 when no conflict where present to neighbors. The neighbors took the minimum of its message and the parent message (t-count). If t-count is superior to the distance of any agent, the algorithm has finished as any constraints have been resolved.