Computational Game Theory (CIS 620/OPIM 952)

Instructor: M. Kearns

Luis Ortiz February 4, 2003

Graphical Games

Part 3: [Dealing with arbitrary topology]

Alternative Formulations and Algorithms

- Computing NE can be formulated as a search over an uncountably infinite set.
  - Want to find a mixed strategy assignment to each player (out of all possible values for mixed strategies) such that each is (approx) best response to the others.

- Discretizing each player's mixed strategy space
  $\Rightarrow$ search over finite space of possible values
  [cross-product of finite set of possible values for each players mixed strategies in the discretization]

- Many ways to perform search.

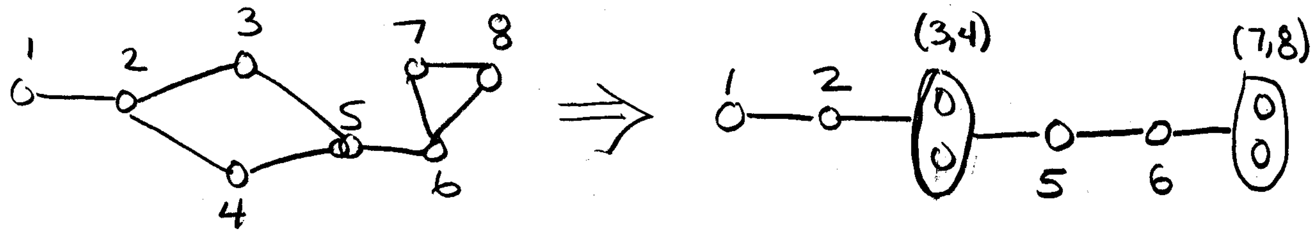
- Nashprop is a particular example: exploits local graph topology.
  [local search with backtracking].

- Other alternatives (including "brute-force" search).

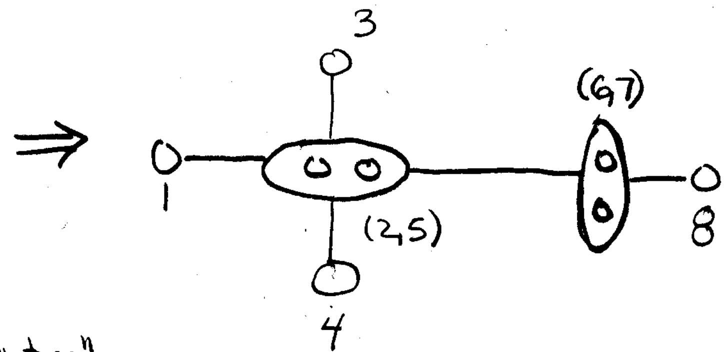
Node-merging: Another heuristic approach to handle graphical games with graph of arbitrary topology.

Idea: Turn graph into a tree by merging nodes

Ex.



Alternative: hyper-tree



Objective: Want "hyper-tree" graph whose size of largest "hyper-neighborhood" is smallest.

[A computationally hard problem!]

Alg: Adapt TreeNash accordingly to work on tree graph with "hyper-nodes"

Remarks:

- "Hyper-table" size is exponential on size of largest pair of neighboring "hyper-nodes"
- Running time is exponential on size of largest "hyper-neighborhood".

Compare and contrast: Nashprop vs. Node-merging

- Some graphs can lead to trees with large size "hyper-neighborhoods" even if node-merging is optimal!

  ⇒ large amount of computation/storage can be required from the start

  Once, "downstream" (table) pass performed, (possibly hard) "upstream" (assignment) pass easy (requires no backtracking!)

- Nashprop works directly on the game graph. Each round can be done in poly. time space/time in game representation size (as long as max. neighborhood size doesn't grow to fast with number of players).

  However, assignment-passing phase can take a large # of rounds (exponential in # players).

- A function minimization approach (VK'02)

Idea: Apply greedy (coordinate-wise) "hill-climbing" method to minimize a particular "cost" function whose global minima correspond exactly to Nash equilibria.

Ex. of a cost function

Given a joint mixed strategy, the cost function outputs the total sum of each player's regret (the amount a player can gain for itself by unilaterally deviating from its assigned mixed strategy in the given joint mixed strategy).

Defn of gain function

For a given joint mixed strategy, the gain wrt a player is the largest reduction in cost achievable by unilaterally deviating from the strategy assigned to the player in the joint mixed strategy.

[Note: changing a player's mixed strategy "affects" both the player's regret and that of its neighbors!]

Alg. sketch:

For each round,

let i^{max} be player with largest gain given the current joint mixed strategy.

If the gain wrt player i^{max} is positive.
Set player i^{max} current mixed strategy to be that which maximizes the gain w.r.t. player i^{max}.

Remark: • "regrets" are continuous but not differentiable, yet...

• For a graphical game,
Both the gain of each player and the mixed strategies that achieves that gain can be computed efficiently by a linear program whose variables "correspond" to the player's mixed strategy and its neighbors' expected payoff.

[Intuition: In a graphical game, a player's mixed strategy "affects" only its expected payoff and that of its neighbors in the game graph].

• Only convergence to local optima guaranteed. So it might not find a NE [i.e., incomplete search].

- Computational considerations of "hill-climbing" heuristic.
  - it might take a long time to converge (find a local minimum), but...
  - each iteration (mixed-strategy update) can be done efficiently (poly. time in model size).
  - little space considerations.

Computing NE: Constraint Satisfaction Problem (CSP) formulation:

- A CSP

<u>Input</u>

- A set $\{P_i\}$ of n variables
- A set $\{D_i\}$ of n domains (possible values for vars.)
- A set $\{C_j\}$ of m constraints (functions from $D_1 \times \cdots \times D_n$ to $\{0,1\}$)

<u>Output</u>

- An assignment $\vec{p}^* = (p_1^*, \ldots, p_n^*)$ to the variables $\vec{P} = (P_1, \ldots, P_n)$ s.t.

$$p_i^* \in D_i \quad \forall i = 1, \ldots, n$$

$$C_j(\vec{p}) = 1 \quad \forall j = 1, \ldots, m$$

<u>Remarks</u>:

- Often the domains are finite sets
- Often the constraints are over a cross-product of a subset of the domains (as opposed to all the domains): each constraint is a function of only some subset of the vars.

For computing a NE.

P_i corresponds to player i's mixed strategy.

D_i " " mixed strategy space (set of possible values/mixed strategies that player i can play).

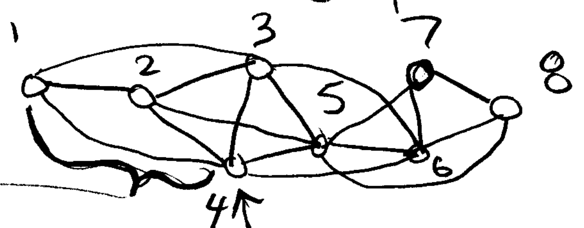
· We have one constraint per player

$C_i(P_1, .., P_n)$ corresponds to the best-response condition for player i.

<u>Remarks</u>:

- If discretization used, D_i are finite sets (corresponding to the discretised points)
- If approx. NE, C_i correspond to <u>approx</u> best-response condition for player i.

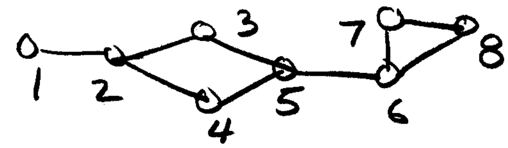
<u>Constraint network</u>: graphical "representation" for a CSP.



each node corresponds to a variable in the CSP.

each edge ⇒ ∃ a constraint that is a function of (at least) those variables.

Example: Gb:



Vars: $\{P_1, \dots, P_8\}$

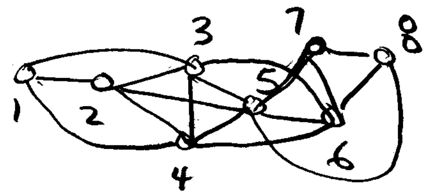
Domains: $\{\underbrace{D, \dots, D}_{8}\}$

Constraints: $\{C_1, \dots, C_8\}$

$\uparrow$

(ε)-best-response constraints for player 1, etc...

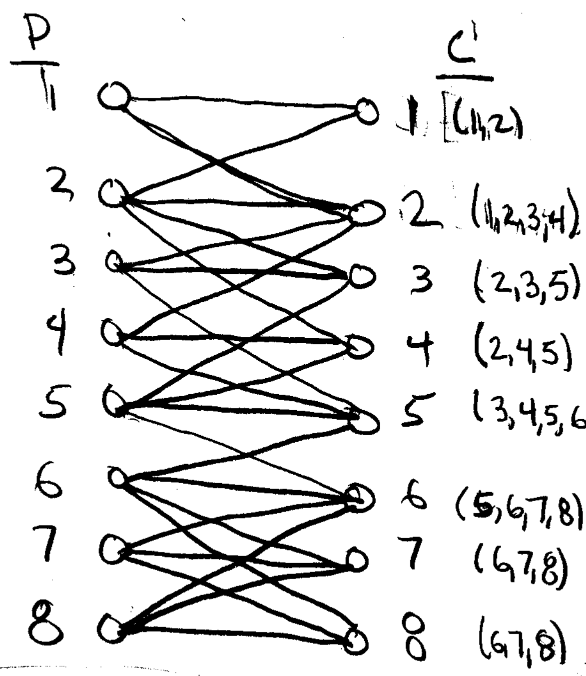
Constraint network:



Alt. CSP formulations:

① Hidden-variable [constraints are hidden vars; new CSP has only binary constraints]

Can be simplified by first merging constraints.



Vars: $\{P_1, \dots, P_8, C_1', \dots, C_8'\}$

Domains: $\{D, \dots, D, D_1', \dots, D_8'\}$

$D_1' = \{(P_1, P_2) \in D^2 : C_1(P_1, P_2) = 1\}$

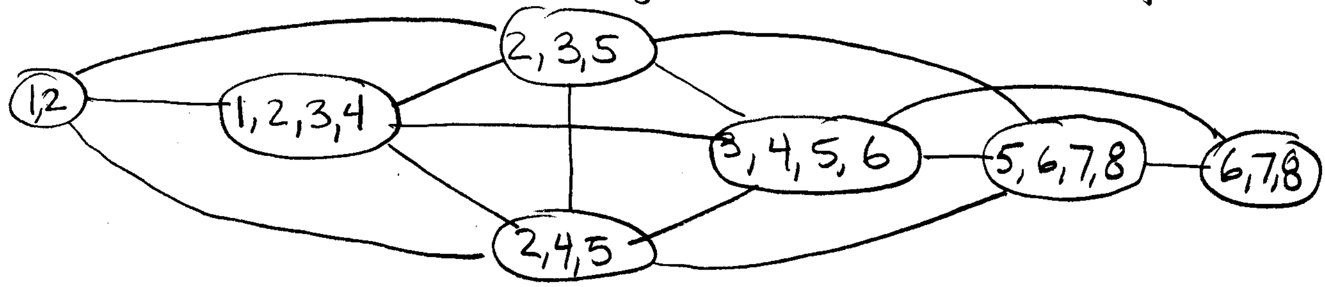
= "Set of consistent assignments of vars. for constraint 1" (etc.)

Constraints: $\{C^{new}_{(i,j)} : C_j$ is a function of (at least) P_i in orig. CSP$\}$

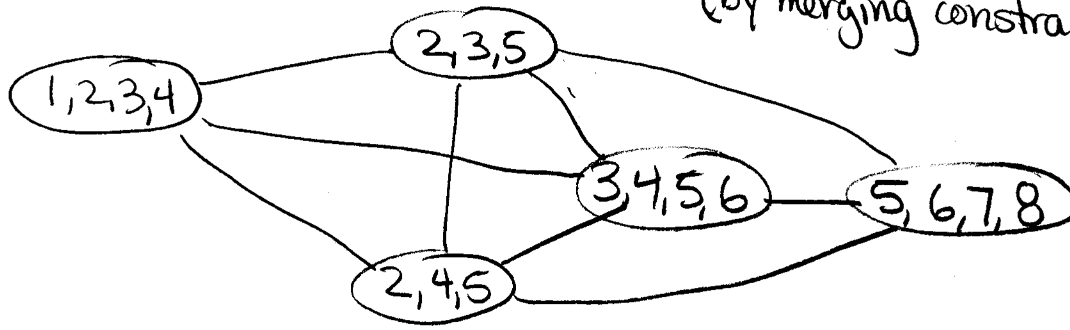
Ex: ~~Denote~~ $C^{new}_{(1,2)}(P_1, \vec{C}_2') = 1$ iff P_1 and $\vec{C}_2'$ consistent (i.e., $P_1 = P_1'$)

(equiv to) iff $C_2(P_1', P_2', P_3', P_4') = 1$ and $P_1' = P_1$

② Dual [vars. are constraints of orig. CSP.; new CSP has only binary constraints]



⇓ Can be simplified to (by merging constraints)



Vars: $\{ P^{dual}_{1234}, P^{dual}_{235}, P^{dual}_{245}, \dots \}$

Domains: $\{ D^{dual}_{1234}, D^{dual}_{235}, \dots \}$

$D^{dual}_{1234} = \{ (P_1, P_2, P_3, P_4) \in D^4 : C_1(P_1, P_2) = 1, C_2(P_1, P_2, P_3, P_4) = 1 \}$

(etc.)

Constraints: $\{ C^{dual}_{(1234, 235)}, \dots \}$

Ex: Denote $\vec{P}_{1234} \equiv (P_1', P_2', P_3', P_4')$, $\vec{P}_{235} \equiv (P_2'', P_3'', P_5'')$

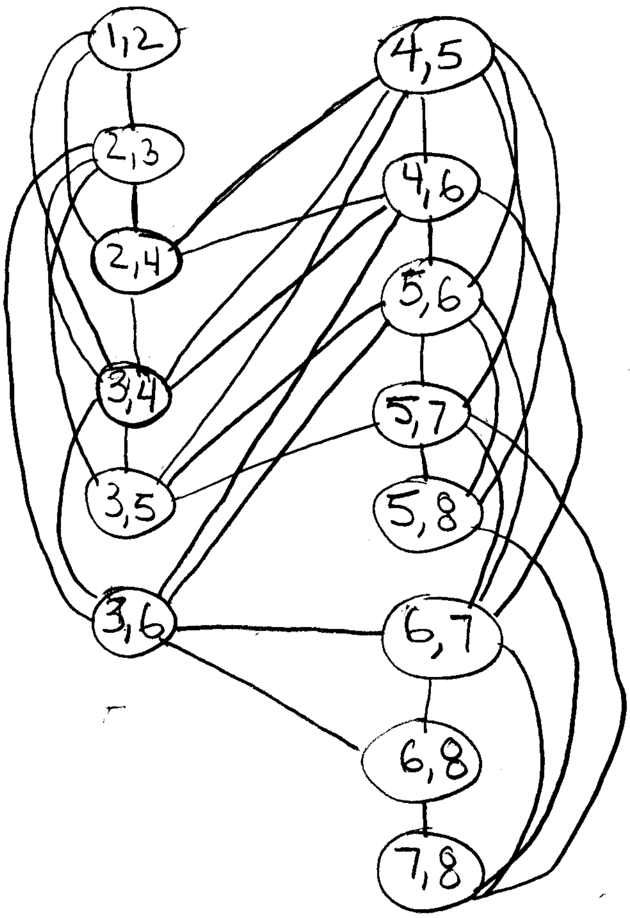
$C^{dual}_{(1234,235)}(\vec{P}_{1234}, \vec{P}_{235}) = 1$ iff $\vec{P}_{1234}$ and $\vec{P}_{235}$ consistent (i.e., $P_2' = P_2'', P_3' = P_3''$)

[assignment corresponding to intersection are consistent]

iff $C_1(P_1', P_2') = 1, C_2(\vec{P}_{1234}) = 1, C_3(\vec{P}_{235}) = 1$ and $P_2' = P_2'', P_3' = P_3''$

③ Clusters of variables:

Ex, a pair of variables $(P_i, P_j) \equiv (P_j, P_i)$ is a new variable if $\exists$ a constraint that is a function of P_i, P_j



- Rewrite constraints as a function on such pairs of vars.
- So now two nodes in the new CN has an edge if the orig. vars. from which those nodes where form are all vars. of some orig. constraint.

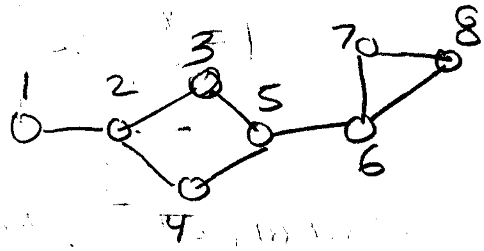
Ex: Denote $\vec{P}_{23} \equiv (P_2, P_3)$, $\vec{P}_{35} \equiv (P'_3, P'_5)$, $\vec{P}_{25} = (P''_2, P''_5)$.

$C_3^{new}(\vec{P}_{23}, \vec{P}_{25}, \vec{P}_{35}) = 1$ iff $C_3(P_2, P_3, P_5) = 1$
and $P_2 = P''_2$
$P_3 = P'_3$
$P'_5 = P''_5$
iff $\vec{P}_{23}, \vec{P}_{25}, \vec{P}_{35}$ consistent among themselves and with constraint

- Can use clusters of larger or diffrent sizes!

Variable elimination: A general "algorithm"

- Here we apply it to solve CSPs.
- Consider the graphical game (GG) example:



Observation:

- Let the "cost" function $S(\vec{p}) = \max_{i=1,\dots,n} L_i(\vec{p})$.

where $L_i(\vec{p})$ is some "loss function" associated with player i.

- If we let $L_i(\vec{p}) = \begin{cases} 0 & \text{if } \vec{p} \text{ "satisfies" (best response condition for player } i \\ 1 & \text{otherwise.} \end{cases}$

Then $\vec{p}$ is a (ϵ)-NE for the game iff $S(\vec{p}) = 0$.
iff $\vec{p}$ is global min of $S(\vec{p})$

- To find Nash equilibria in space of mixed strategies, we want to find $\vec{p}^*$ which globally minimizes the "cost" function S:
  - $\vec{p}^* \in \arg\min_{\vec{p} \in D^n} S(\vec{p})$

- In the GG example above, we have,

$$S(\vec{p}) = \max(L_1(p_1, p_2), L_2(p_2, p_1, p_3, p_4), L_3(p_3, p_2, p_5), L_4(p_4, p_2, p_5), L_5(p_5, p_3, p_4, p_6), L_6(p_6, p_5, p_7, p_8), L_7(p_7, p_6, p_8), L_8(p_8, p_6, p_7))$$

$$\min_{\vec{p}\in D^n} S(\vec{p}) = \min_{\vec{p}\in D^n} \max\Big[\Big[L_1(p_1,p_2),\, L_2(p_2,p_1,p_3,p_4),\, L_3(p_3,p_2,p_5),$$
$$L_4(p_4,p_2,p_5),\, L_5(p_5,p_3,p_4,p_6),$$
$$L_6(p_6,p_5,p_7,p_8),\, L_7(p_7,p_6,p_8),\, L_8(p_8,p_6,p_7)\Big]$$

("distribute minimization" w.r.t p_8)

$$= \min_{(p_1,\ldots,p_7)\in D^{n-1}} \max\Big[L_1(p_1,p_2),\, L_2(p_2,p_1,p_3,p_4),\, L_3(p_3,p_2,p_5),$$
$$L_4(p_4,p_2,p_5),\, L_5(p_5,p_3,p_4,p_6),$$
$$\min_{p_8\in D} \max\big[L_6(p_6,p_5,p_7,p_8),\, L_7(p_7,p_6,p_8),\, L_8(p_8,p_6,p_7)\big]\Big]$$

$$=$$

$\vdots$ (continue "distributing minimizations" w.r.t. other vars...)

$\vdots$

$$
\begin{aligned}
\min_{\vec{p} \in D^n} S(\vec{p}) = \min_{p_1 \in D} \min_{p_2} \max\Big[& L_1(p_1, p_2), \min_{p_3} \min_{p_4} \max\big(L_2(p_2, p_1, p_3, p_4), \\
& \min_{p_5} \max\big(L_3(p_3, p_2, p_5), L_4(p_4, p_2, p_5), \\
& \min_{p_6} \max\big(L_5(p_5, p_3, p_4, p_6), \\
& \underbrace{\min_{p_7} \underbrace{\min_{p_8} \max\big(L_6(p_6, p_5, p_7, p_8), L_7(p_7, p_6, p_8), L_8(p_8, p_6, p_7)\big)}_{L_9(p_5, p_6, p_7)}}_{L_{10}(p_5, p_6)} \big)\big)\big)\Big]
\end{aligned}
$$

etc. . . .

<u>High level picture</u>:

Can decompose (optimization) problem into many (hopefully) <u>smaller</u> subproblems!

[By exploiting <u>structure</u> in the original problem]

Cost Minimization Problem (CMP)

• **Motivation**: "Select smallest possible ε (regret) achievable for a **given** discretization size τ"
[or smallest ~~of~~ largest "regret" obtained].

• Let $L_i(\vec{p})$ = "regret for player i"

$$\left[\equiv \max_{a} M_i(\vec{p}[i:a]) - M_i(\vec{p}) \right]$$

• Same operation as for (ε)best-response CSP (except different "loss functions")

• For "loss functions" just defined,

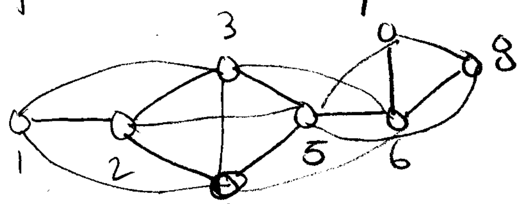
$$\min_{\vec{p} \in D^n} S(\vec{p}) = \text{"smallest possible regret } \varepsilon \text{ achievable for given } \tau\text{"}$$

$$= \min_{\vec{p} \in D^n} \max_{i=1,..,n} L_i(\vec{p}).$$

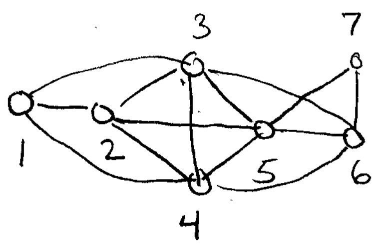
Variable elimination

• Can be seen as operation on same graph as constraint network (CN)

For example,



$\Downarrow$ "eliminating variable p_8"



$\Downarrow$ so on.

- In general, "eliminating" one variable requires the following graph operation:

  1. for every pair of neighbors of variable to be eliminated, add an edge between those neighbors (if one does not already exist).

  2. remove variable (node) from graph and all edges incident to it.

  Perform same operation recursively.

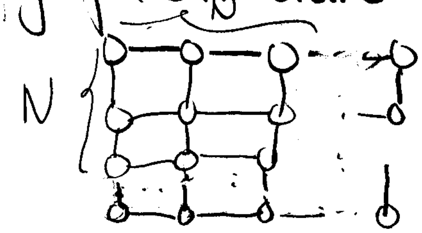
- Remarks:

  Graph terminology: A clique is a set of nodes that are "fully connected" in the graph (every pair nodes in the set has an edge joining them).

- Space & time is exponential in the size of the largest clique found in any of the graphs obtained by the process above. (depends on elimination order).

- In GG example above, size of largest clique found for particular elimination order is 4, which in this case is equal to the max neighborhood of the graph of the game (so running time/space is poly. in model size for this example).

  In general, this won't be the case. Consider for example (NxN) grid graph structure



Remarks:

- Gaussian elimination (for solving linear systems of equations) is a special case of variable elimination.

- Tree Nash can also be seen as a special case of variable elimination

- In particular, for GGs with arbitrary graphs, Variable elimination computationally equivalent(*) to applying TreeNash to "hyper-tree" resulting from node-merging operation.

(*) Any resulting "hyper-tree" has a (not necessarilly unique) "corresponding"/or "associated" elimination order ~~for~~ for which var. elim. running time is of the same order as running Tree Nash in the "hyper-tree"; and vice versa...

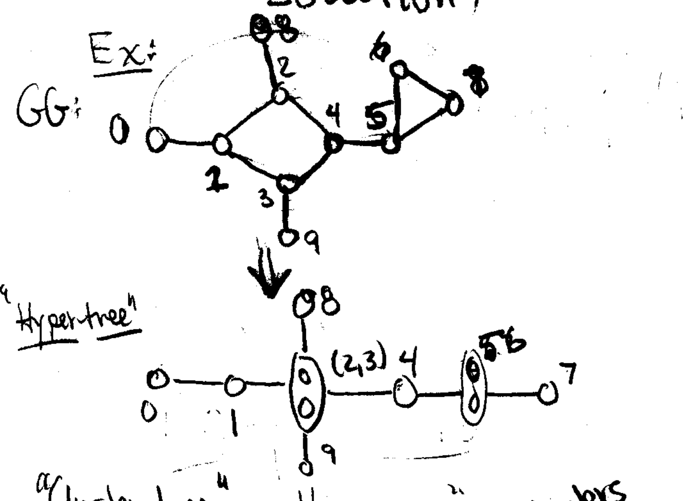
Hybrid approches: (UK'02) [A brief sketch]

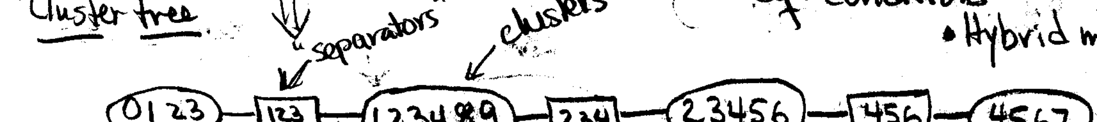
- Approx eq. refinement:

  Idea: perform "search" in a small subset of the space; recursively, make finer disc**retization**.

  - Watch out! Might "lose" eq. in the process. (An εNE in some region of the space does not guarantee that ∃ exact NE in that region; finer granularity in the disretization of a region might not lead to Nash equilibrium)

- Subgame decomposition: An alternative representation/formalization of the CSP (closely connected to the process of variable elimination used to compute a solution)





- Assign a player, (eq. condition) to exactly one cluster s.t. original neighbors also in cluster [ex: assign 2,3,8,9 to cluster (123489)]
- Idea: "Clamping" mixed strategies for players in separator [123] and [234] render eq. in cluster (123489) conditionally indep. of eq. in other cluster. [i.e., values for 8,9 that satisfy eq. cond. given values for {1234} = [123] ∪ [234] assuming values for separators consistent!]
- Set up a CSP where separators are vars and (binary) constraint among them reflect consistency and eq. conditions.
  - Hybrid method results from using "Hill climbing" within cluster ~~constraints~~ (as opposed to using var. elim.).

- Other algorithms exist for solving CSP.
- Constraint propagation algorithms: CP, based on the notion of arc consistency (AC)
- NashProp can be seen as a particular instantiation of a constraint propagation alg. for AC.