Distributed Problem Solving and Planning

Edmund H. Durfee
University of Michigan, USA

Goals for this Talk

- Introduction to a variety of concepts, issues, and techniques
- Mostly emphasis on improving your awareness – you won't be experts (yet)
- Dig down in a few places into some details to give you a flavor of operationalization
- Examine some research issues in a little more depth to explore some strategies for extending the state of the art

How Does DPS Relate to MAS?

Multi-Agent System: Emphasis on the fact that there are multiple agents, leading to concern about intrinsic properties such as: truth-revelation, manipulation, coherence, Pareto efficiency, …

Distributed Problem Solving: Emphasis is on solving one or more problems, through efforts of multiple agents, with concerns about extrinsic properties such as competence, robustness, resource efficiency, distraction, …

What is "Problem Solving"?

Search through a state (solution) space

Begin at an initial state (partial or empty solution)

Apply operators to states to generate successor states

Find a state (solution) that satisfies a goal test

Return state (solution) and/or path from initial to solution state.

What Changes with “Distributed”?

- Different portions of the state (solution) space are known to different agents
- A “state” could be distributedly defined
- Agents might have different operators to apply to (partial) states to generate successor states
- Agents might have different goal tests
- Solution state and/or path might require composition from multiple agents

Example 1: Hidden Pictures

- **Simple (visual) search task**
- **How would <u>you</u> work as part of a team to solve it?**

Example 1: Hidden Pictures

Example 1: Discussion

Decomposition into independent subproblems (search areas or objects)

Allocation of subproblems to team members

Pursuit of subproblem solutions

Overall solution synthesis:

- Simple! When object is found by any member, the team knows it (if the member tells them!)
- Someone needs to keep track of which tasks are done and which still need to be done.

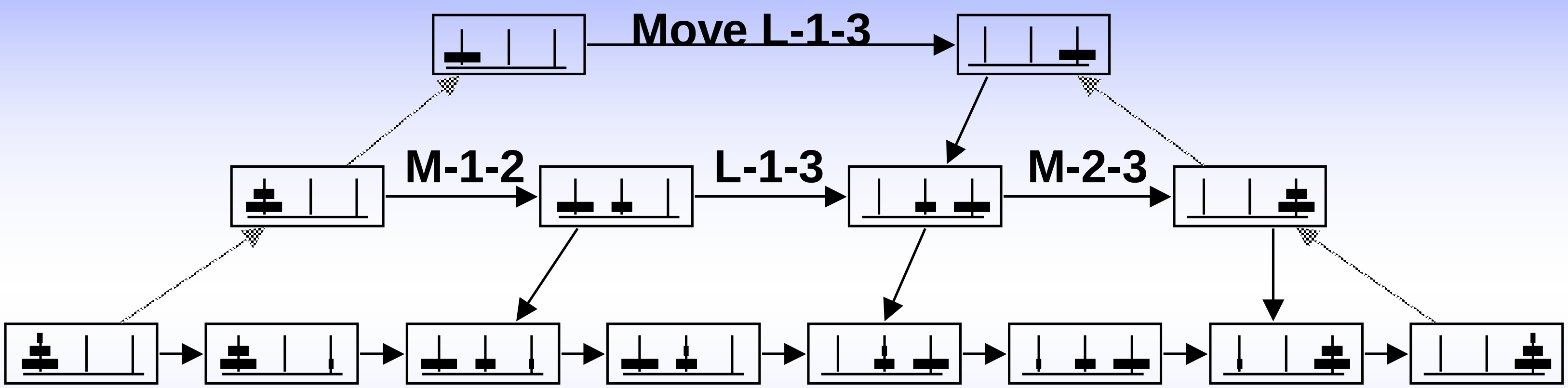
Homogeneous Agents

For some applications, agents are identical in their knowledge/expertise/capabilities

DPS with homogeneous agents therefore simply amounts to:

- Decomposing larger tasks into smaller tasks of the same kind
- Assigning smaller tasks
- Solving smaller tasks (possibly requiring recursive decomposition and assignment)
- Recomposing the results

A Canonical Example



Initial Problem

- **n** length of solution
- **`f(n)`** complexity of search
- **k** ratio of solution lengths between levels

Search Reduction

Single-Level Abstraction Hierarchy

- Time (& Space) Complexity $O(\sqrt{n}\, f(\sqrt{n}))$
- For example: $b^n \Rightarrow \sqrt{n}\, b^{\sqrt{n}}$ and $n^3 \Rightarrow n^2$

Single-Level, Multiple Agents

- Solve subproblems in parallel
- Time Complexity $O(f(\sqrt{n}))$
- For example: $b^n \Rightarrow b^{\sqrt{n}}$ and $n^3 \Rightarrow n\sqrt{n}$

Search Reduction (cont.)

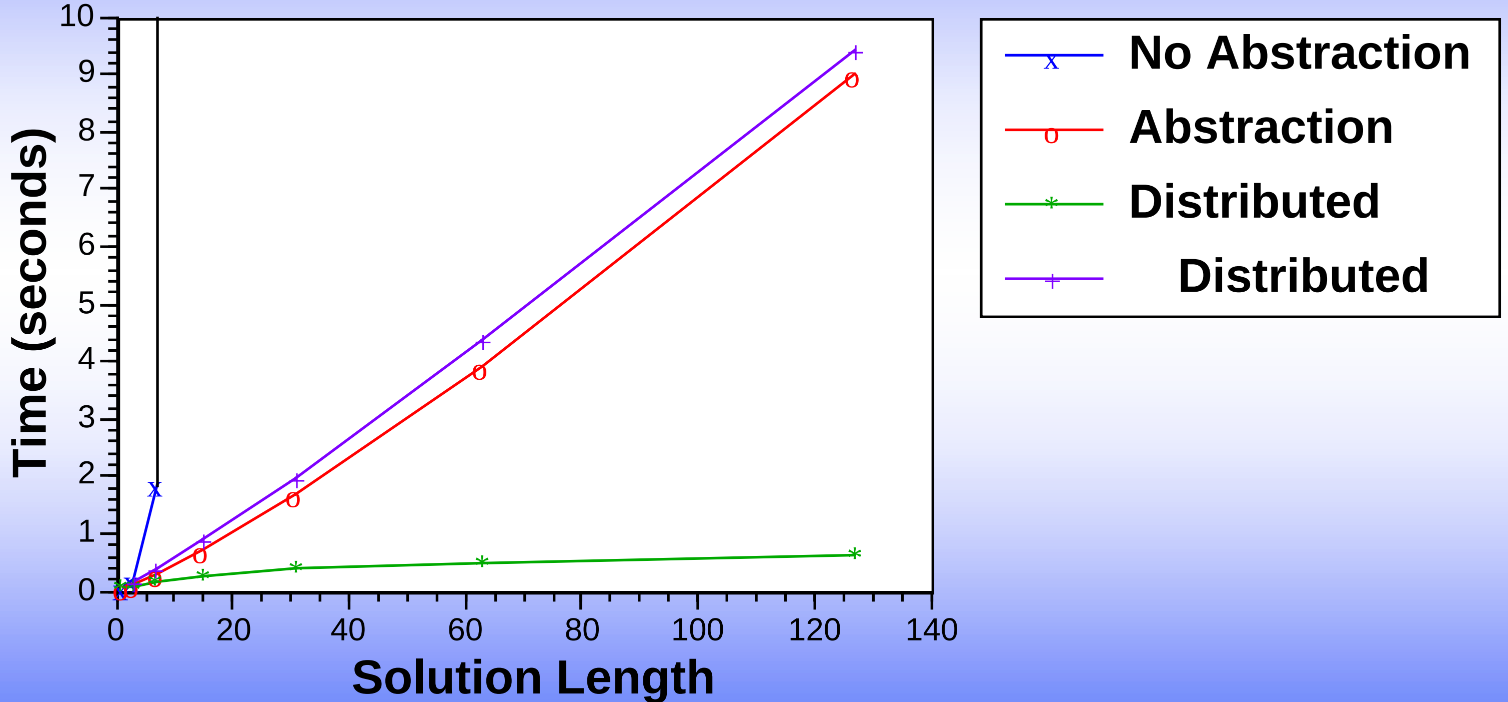
Multi-Level Abstraction

- l levels of abstraction
- Time (& Space) Complexity $O(n)$
- l must grow with n ($l = \log_k n$)

Multi-Level, Multiple Agents

- Time Complexity $O(\log_k n)$
- l and number of agents must grow with n

Towers of Hanoi
Experimental Results



Heterogeneous Agents

For many applications, agents have complementary knowledge/capabilities

Challenge lies in matching tasks with agents that can carry them out

Strategies for doing so:

- Matchmaking
- Brokering: A broker for a particular kind of task accepts it and selects which agent will actually carry it out
- Contracting

Contract Net Protocol

1. Agent with a task decomposes it into subtasks to be contracted out
2. That *manager* announces the subtask
3. *Contractor* agents that are eligible to bid can submit bids
4. After enough time has elapsed, manager chooses from among submitted bids and makes one or more awards
5. Winning contractors provide interim and final reports of subtask accomplishment

Contract Net Protocol 1

1. Agent with a task decomposes it into subtasks to be contracted out
   - Not as simple as you might think!
   - Knowledge of how to decompose (and recompose) must be available to the manager
   - Often, there are alternative decompositions
     - Try them at the same time? Overcommitment?
     - Pick the most promising one? How? Tentative announcements?
     - Permit decommitment?

Contract Net Protocol 2

2. That *manager* announces the subtask

- Announcement includes
  - Eligibility specifications

    Constraints on who is even allowed to bid
  - Bid specifications

    What a bid should tell the manager
  - Timeout when award decision is to be made
- Eligibility might be overconstraining
  - Balance can be difficult: too open means too many bids (and wasted resources); too closed might mean no acceptable bids, so have to retry

Contract Net Protocol 3

3. *Contractor* agents that are eligible to bid can submit bids
   - A “bid” does not necessarily involve (monetary) compensation for services
   - Often assumes available contractors implicitly accrue benefit from awards
   - A bid often specifies how well (timeliness, completeness, confidence, precision) the contractor can accomplish the task

Contract Net Protocol 4

4. After enough time has elapsed, manager chooses from among submitted bids and makes one or more awards
   - How the manager chooses is application dependent
   - If no degrees of “how well” then choose randomly (or lowest cost)
   - If degrees of “how well” then weigh the various factors
   - Multiple awards can occur (for reliability, for example)

Contract Net Protocol 5

5. Winning contractor(s) provide(s) interim and final reports of subtask accomplishment
   - Interim reports can serve as “heartbeats” to reassure manager that subtask is active
   - Interim reports can help manager initiate or redirect activities of other contractors
   - Final report provides subtask result to be synthesized into a complete task result

Contract Net Protocol Issues

Contractors can decompose and manage their subtasks, recursively

Requires a shared ontology to understand each others' tasks, bids, capabilities,…

"Greedy" approach: doesn't look ahead to how current match will affect future match availabilities (decommitment)

Redundant activities: same subtask could arise in multiple subtrees and would be contracted and done redundantly

Contract Net Protocol Variation

Under high task loads, limiting factor is agent availability

Turn Contract Net around:

- Available agent announces what it can do
- Agents with tasks “bid” their tasks
- Available agent accepts the task that it is best suited to do (or that is most critical, or whatever)

Other Task Passing Approaches

Matchmaker: Centralized registry of agent capabilities to be matched with needs

Broker: Accepts all tasks of a particular type, and then assigns agent to do each from a “stable” of capable agents

Supply chain: Contractor bids are “tentative” pending enlisting subcontractors – so entire tree formed before firm commitments are made

Task Passing as Resource Allocation

- Matching agents (which are resources for task accomplishment) to tasks (which consume resources) can be viewed as a supply-and-demand problem
- Market mechanisms can be employed to make the most efficient allocations
  - Assuming a static set of supply/demand
  - Otherwise, "continuously clearing" auction can make fewer guarantees

Example 2: Cooperative Mazes

- Do you search a maze from start to end, or end back to start?
- For cooperative search, you can work bidirectionally in parallel
- Try it!

Example 2: Cooperative Mazes

Example 2: Discussion

Perhaps not as easy as you might have thought?

Issues in "connecting up" the partial paths?

Redirecting partner's search to approach your partial path?

No longer could really do your task without having a sense of the partial result of someone else

Result Sharing

For some problems, task assignment isn't the hard part:

- Might be inherently given based on goals agents are "born" with
- Might be trivially accomplished

However, how a task is done could be very dependent on how other tasks are done!

Performance Measures

Completeness: Amount of the task(s) accomplished

Timeliness: How soon tasks will be completed

Precision: How close to optimal (rather than only satisfactory) the results will be

Confidence: Certainty that task results are (or will be) satisfactory

Distributed Constraint Satisfaction

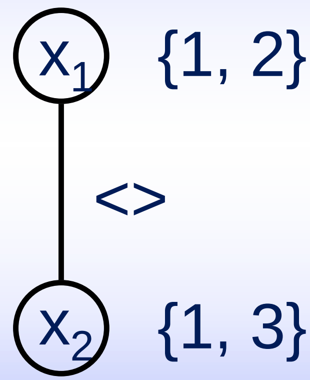
A simple example of result sharing

An agent has the "task" of binding values to one or more variables

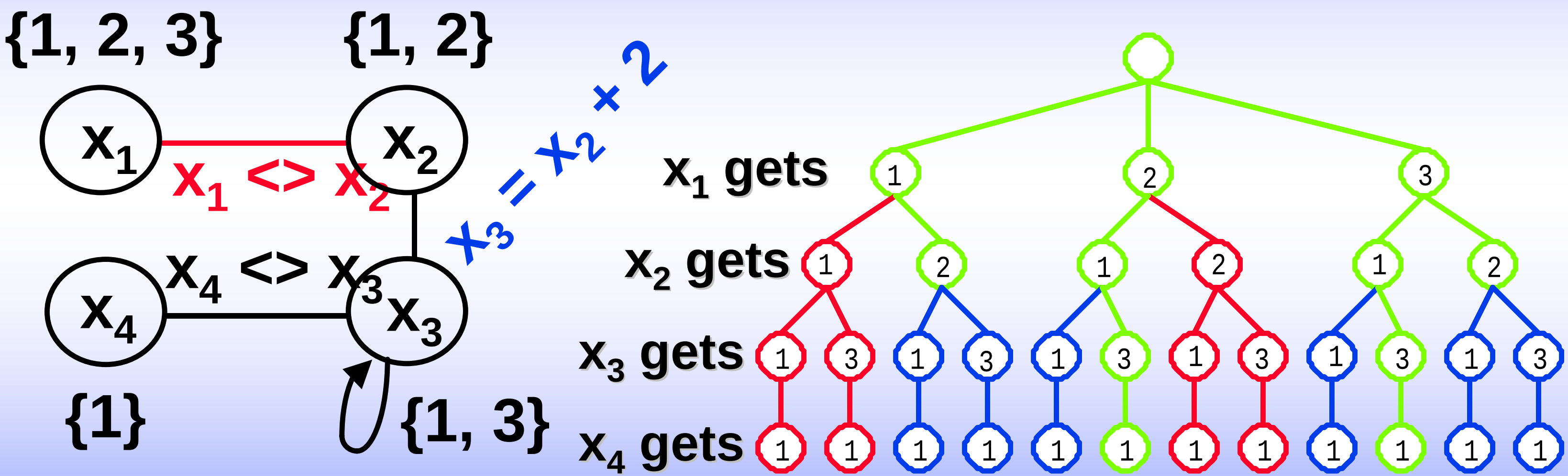
Interdependence arises because of constraints that must hold between the values assigned to variables that are managed by different agents

Constraint Satisfaction

- Variables
- Domains
- Constraints

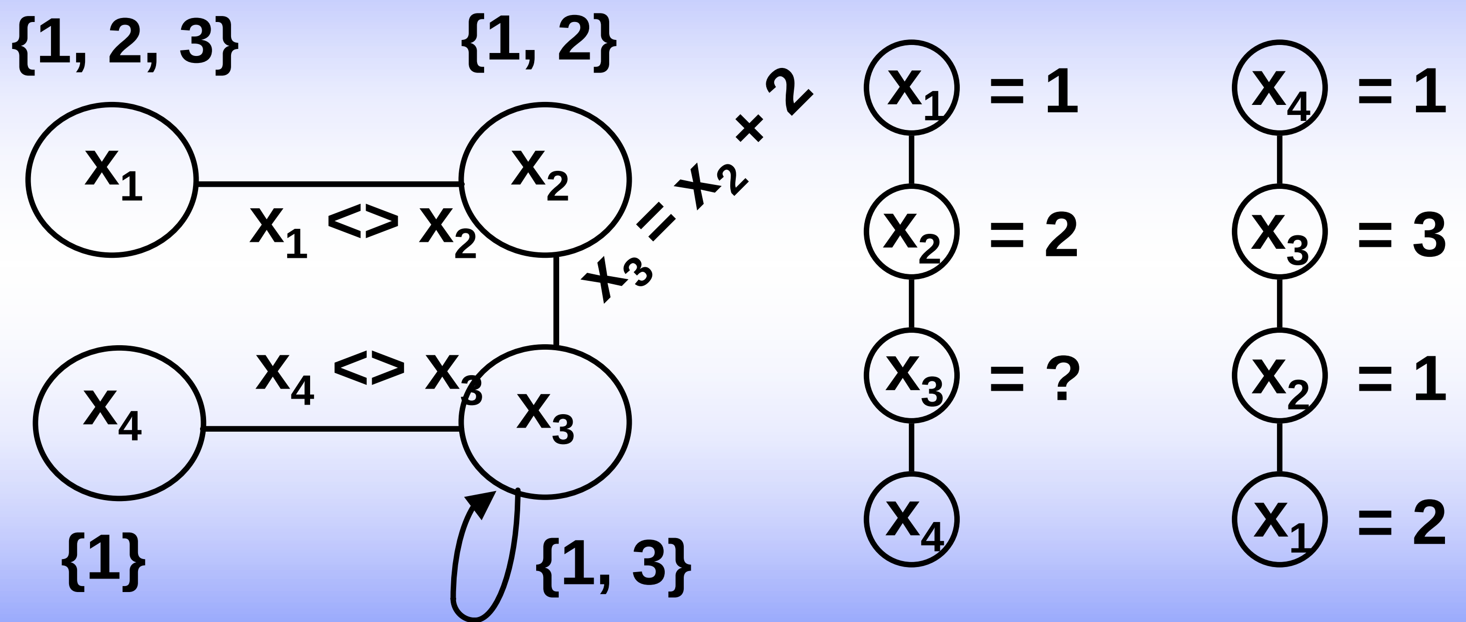


Solving a CSP



Variable Ordering

- **Better Average Case Performance**

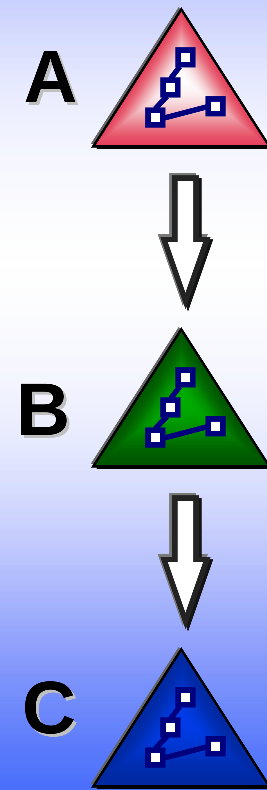


Most Constrained Variable Heuristic

Challenges With Agent Ordering

- A total ordering of the agents controls the concurrent search
  - Information flow
  - Conflict resolution
- The total order greatly affects average search time
- Ordering heuristics aggregate information about variables, domains, and constraints on agents

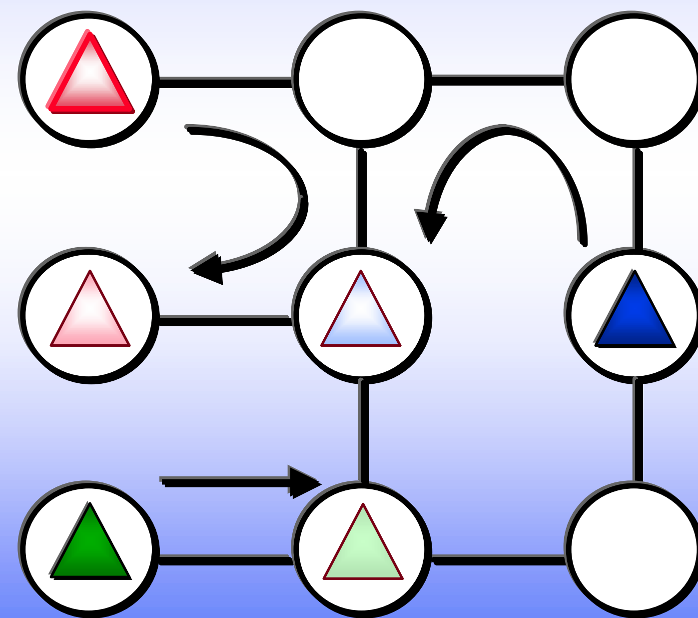
Agents with Variables



Least-Constrained Agent Ordering

- Harder in general since (in CDPS) each agent has many local variables
- Degree of constraint on an agent (its variables) evolves over time as some (combinations of) values are ruled out

Package Delivery Agents



Example DCSP Protocols

(Yokoo)

- Asynchronous Backtracking
  - parallel local CSP solving
  - concurrent, asynchronous, and optimistic passing of bindings downward in agent ordering
  - no-goods passed back up triggering re-search
- Weak Commitment Search
  - agent who discovers a no-good is moved to the top of the agent ordering
  - assumed to reflect more constrained agent

Analysis of Protocols

Advantages

- Parallelism
- Some search for a good ordering

= A_1

= B_3

NG (A_1 B_3)

= B_3

= A_1

NG (A_1 B_3)

Disadvantages

- Fixed ordering strategies
  - ABT: Static
  - WC: Fixed trigger/response
- Potential rediscovery of no-goods due to different orderings

Overall Protocol

- Designated agent initiates an epoch upon start up or when reprioritization conditions met
- Epochs continue until a solution is found or an agent has an empty domain
  - Each agent calculates a local priority measure.
  - The agents form a total order by exchanging priority information.
  - The agents search for a solution using the modified version of asynchronous backtracking.

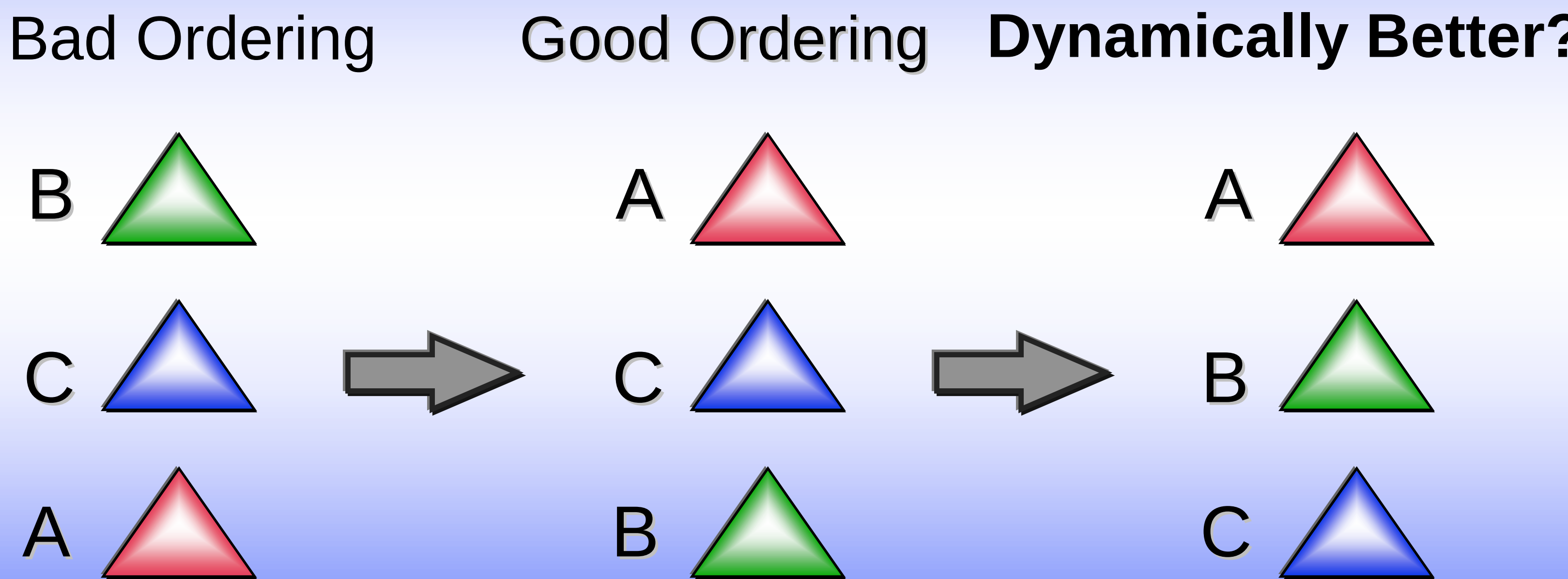
Agent Behavior During Epoch

- Repeat until instructed to halt or reprioritize
  - Pick values for the local variables consistent with the constraints and the (known) values of higher priority agents
  - If the values differ from the previous iteration then alert the next agent in the linear order
  - Else if no values possible then alert the previous agent of a no-good

Reprioritization Conditions

- When "Significant" No-good Discovered
  - Avoid Frequent Restarts
  - The significance approximated by the number of agents involved in the no-good
- Reprioritize when the size of the no-good < *m*, a constant

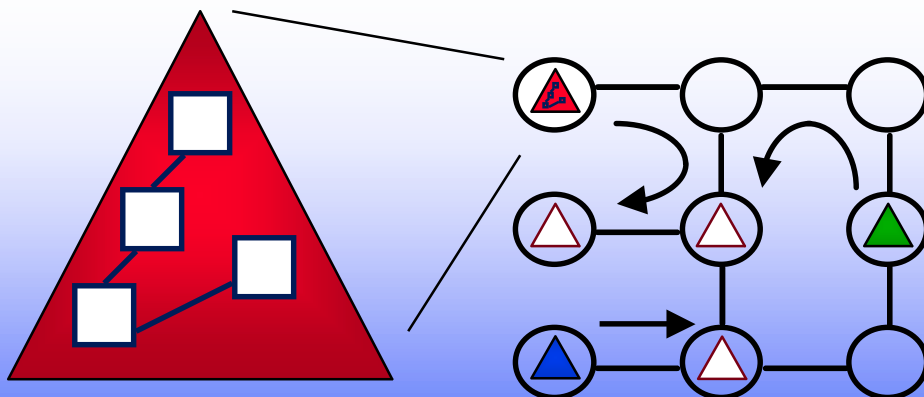
Reordering Between Problem-Solving Epochs



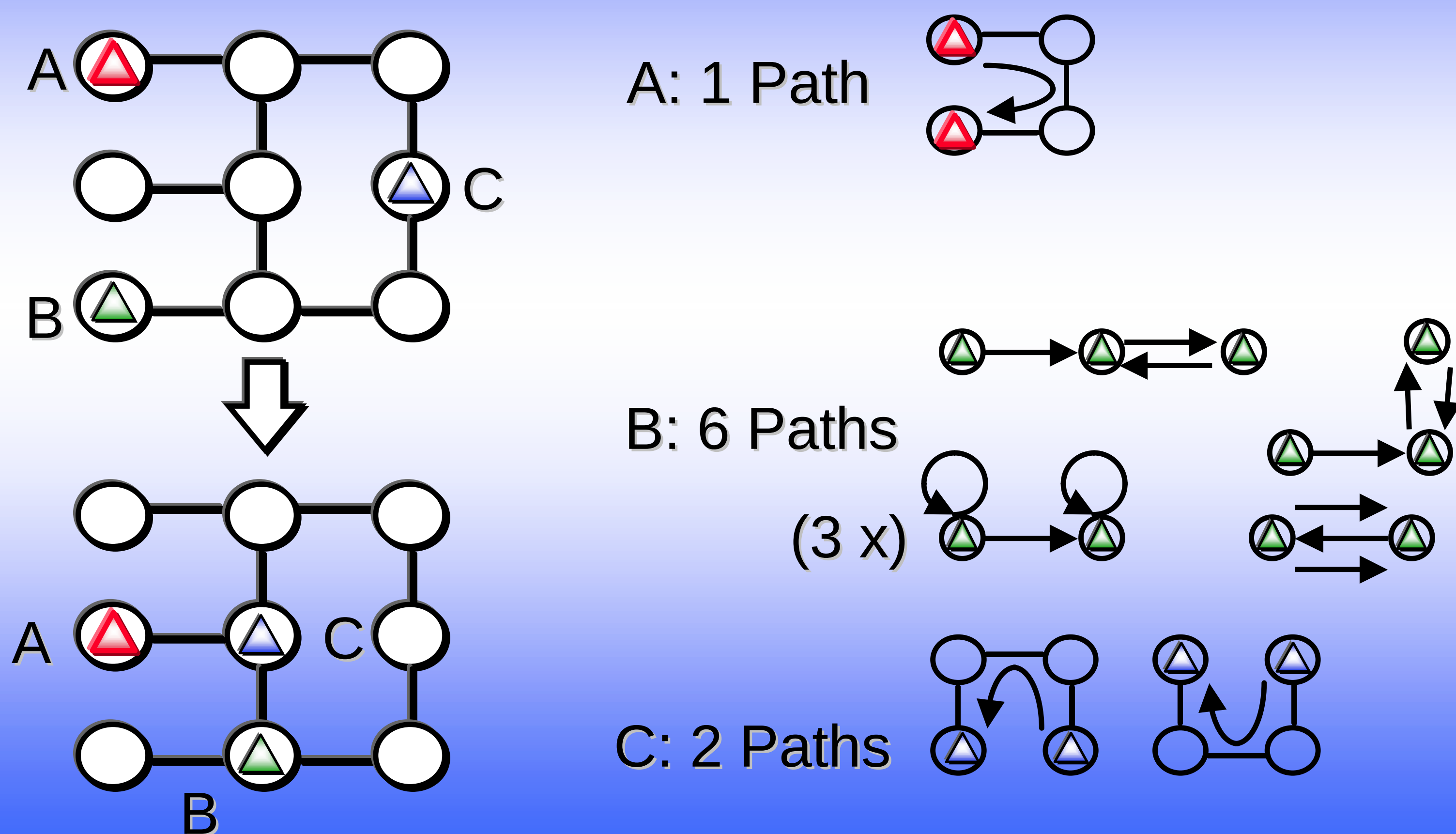
Most Constrained Agent Heuristics

- How Many Local Solutions?

Agent A with Constraint Graph



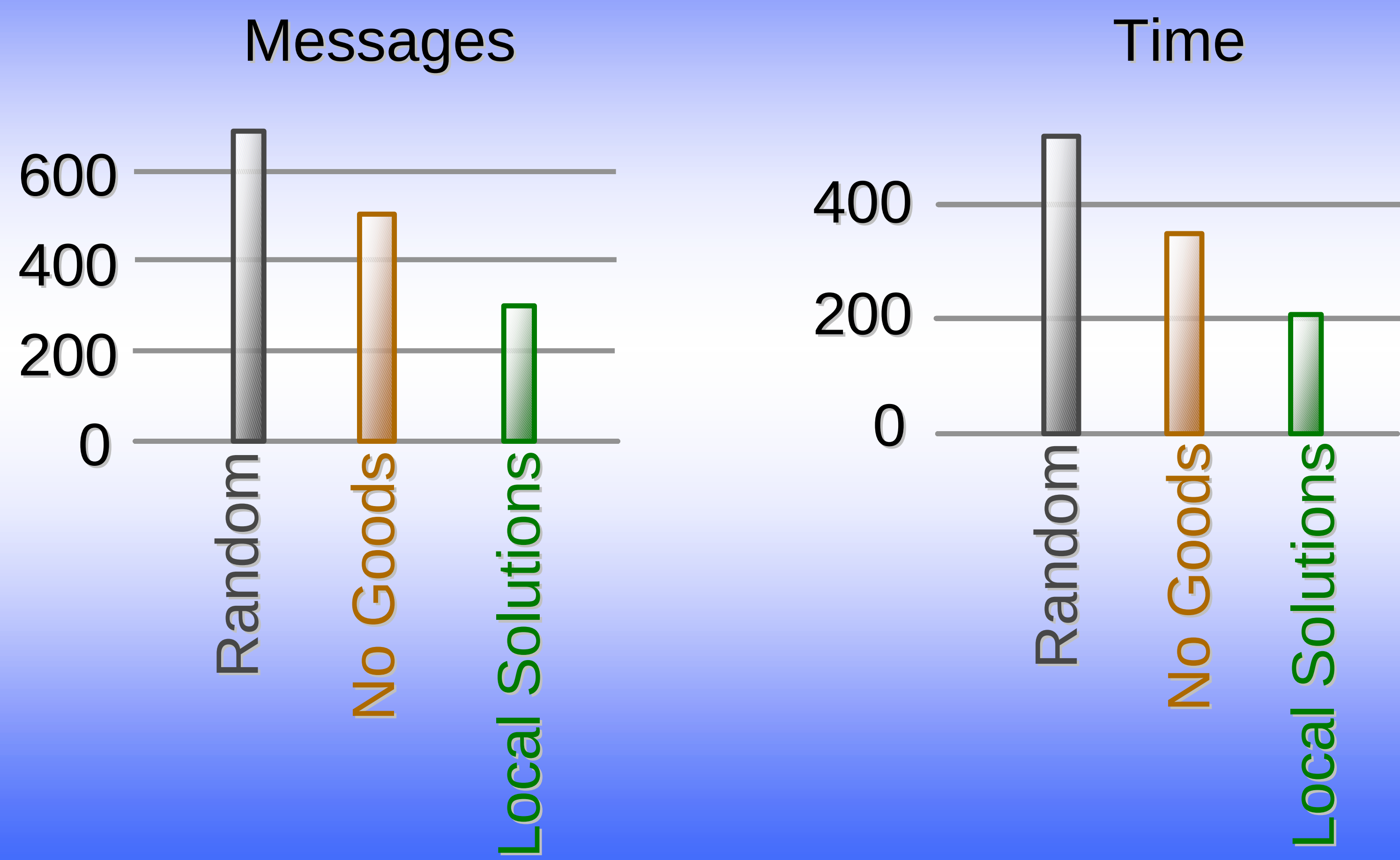
Finding Most Constrained Agents: Reduce agent to a variable



Finding Most Constrained Agents: Approximations

- Use total, average, or weighted average of variable’s domain sizes
- Use number of no-goods discovered or a decaying average of no-goods discovered
  - with exponential decay -> weak commitment
- Search with a Genetic Algorithm

Subset of Results



Evaluation of the Heuristics

- Good orderings had a significant impact.
- The number of local solutions heuristic is best.
- The number of no-goods heuristic worked well.
- The total number of no-goods is more important than a decaying average of the number of no-goods.

DCSP Summary

- Older algorithms were generalized to allow flexible reordering between epochs of (modified) ABT
- Agent-level heuristics for aggregating variable information performed well
- Dynamically acquired information can help

Negotiated Search

What if, as in distributed design, the constraints aren't well-defined at outset, and problem might be overconstrained?

- Use a shared repository (e.g., blackboard) so that decisions made by one agent can be noticed by affected agents
- Permit agents to relax some "constraints"

Now, distributed search involves initiating, extending, and critiquing posted partial solutions, along with relaxation

Constrained Heuristic Search

Essentially, view a variable as having an agent associated with it

Agent receives competing demands for the variable’s value assignment

Agent aggregates demands, and can inform agents that submitted demands of the aggregate demand

Process iterates until demands converge and assignments are made

DCHS for Resource Allocation

1. Each agent has tasks and constraints between them (e.g., ordering)
2. Agent determines how much of which resources it needs and when
3. Agent sends these to “resource” agents
4. Resource tells agents of aggregate demands
5. An agent uses aggregate demands to adjust ordering decisions and resource assignment requests
6. Process can repeat, or an agent might ask resource to commit to a particular assignment
7. If request granted, propagate commitment effects and go to step 2; else change request

Auctions for Resource Allocation

Enough said?

Functionally Accurate, Cooperative

- Consider extreme case: which partial results should be pieced together and how is entirely unpredictable!
- In that case, each agent should share with all other agents each of its partial results
- Eventually, pieces will come together at right agents to be combined into larger and larger results
- Much effort could be wasted: functionally accurate means overall outcome achieves outcome but individual activities might be unproductive
- Agents cannot act independently: cooperative means that co-routining is crucial

Improving FAC

“Each agent should share with all other agents each of its partial results” can be far too costly!

Avoid sharing every single partial result – but what if a crucial one isn’t shared?

Avoid sharing partial results with all other agents – but what if the right agent doesn’t get it?

Communication Strategies

Sending too many partial results can consume bandwidth and computational resources, and can distract recipients into unproductive efforts, such as duplicating search going on elsewhere

Chicken-and-egg problem: Hard to know if a partial result is useful until it is sent!

Role of partial result:

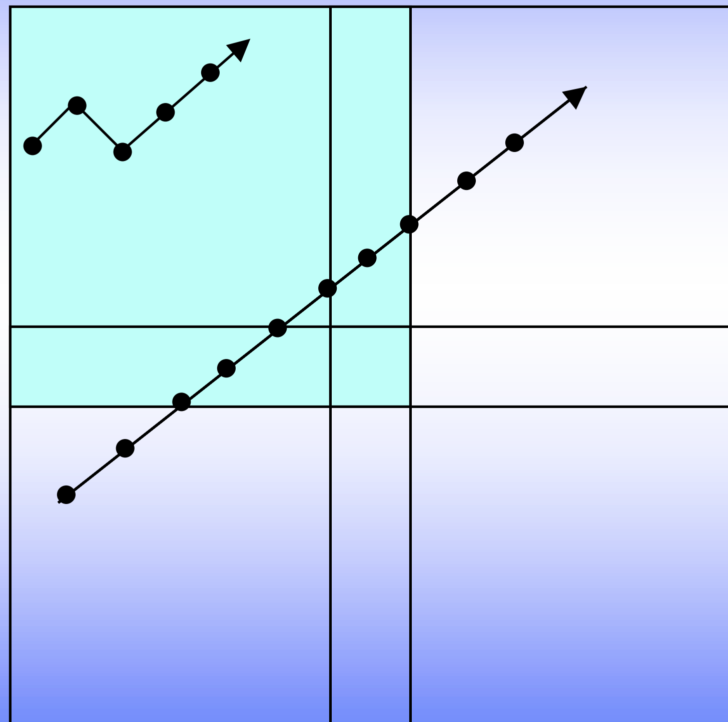
- Contribute to solution: in that case, wait until the whole partial result is done before sending
- Redirect other agents: in that case, send early piece to provide guidance

Example: Distributed Theorem Proving

Organizational Structuring

- If agents know something about "interests" or responsibilities of others, can avoid sending them uninteresting messages
- Knowledge of local organization can be used by an agent to target messages
- In simple form, have templates for other agents: if a partial result matches its template, then send it to the agent
- Richer forms to prioritize communication and processing, and even meta-level!

OS Example:
Distributed Vehicle Monitoring



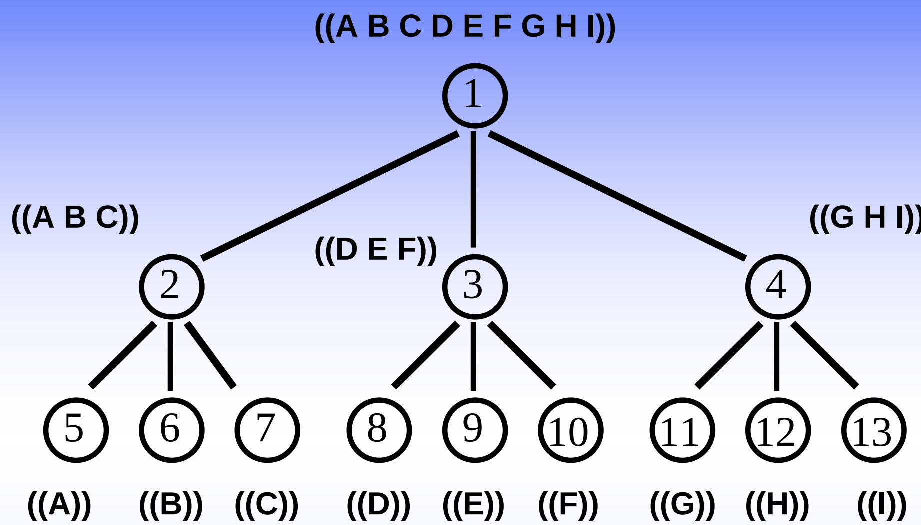
Organization Design

Given that what is desired is a combination of static roles/guidance (called the organization)..

... and runtime coordination mechanisms to revise/refine organizational structure...

... how do we analyze/predict the performance of such combinations to codesign the organization and the agents' coordination mechanisms.

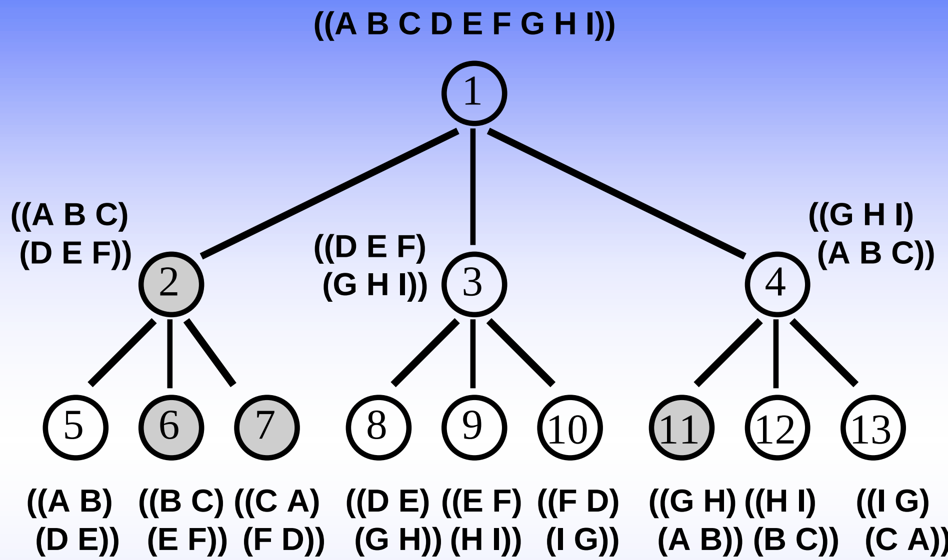
Multilevel Hierarchical Organization



- *N* tasks (in example, 9: A-I)
- *k* subordinates (in example, 3)
- *m* tasks per role at leaves (in example, 1)

Task results passed back up and synthesized; A-I must all be done.

Organization Reliability Through Redundancy



- *o* out of *k* subordinates can fail (example, 1)
- potential duplication of effort (e.g., D, E, F, I)
- delays in completing tasks (in example, A)

Purposes of Runtime Coordination

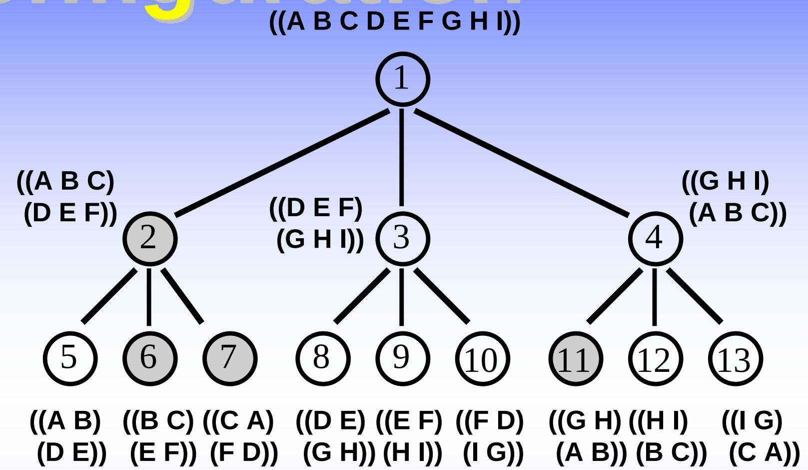
- When agents don’t fail, they should be working on complementary tasks
  - less coordination needed if no redundancy
- When agents do fail, the remaining agents should occupy the most important roles
  - less coordination needed if total redundancy

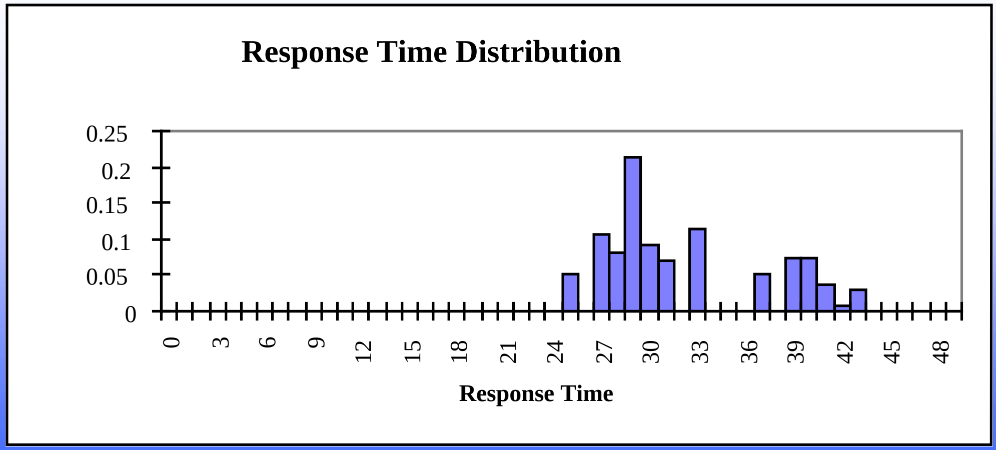
Performance Measures: Response Time

- Besides *N*, *k*, *m*, and *o*, parameters are
  - γ = task execution time / comm delay (assumed to be 1 here)
  - *s* = coordination strategy
  - *f* = agent failure probability (assumed constant and independent here)
- Response time also depends on:
  - which agents fail
  - what the others do

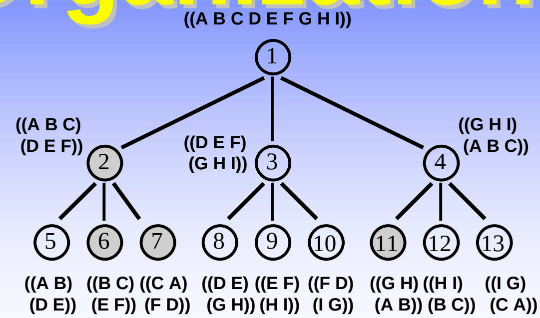
Response Distribution for a Configuration

- For a particular configuration of failed agents, response depends on task ordering

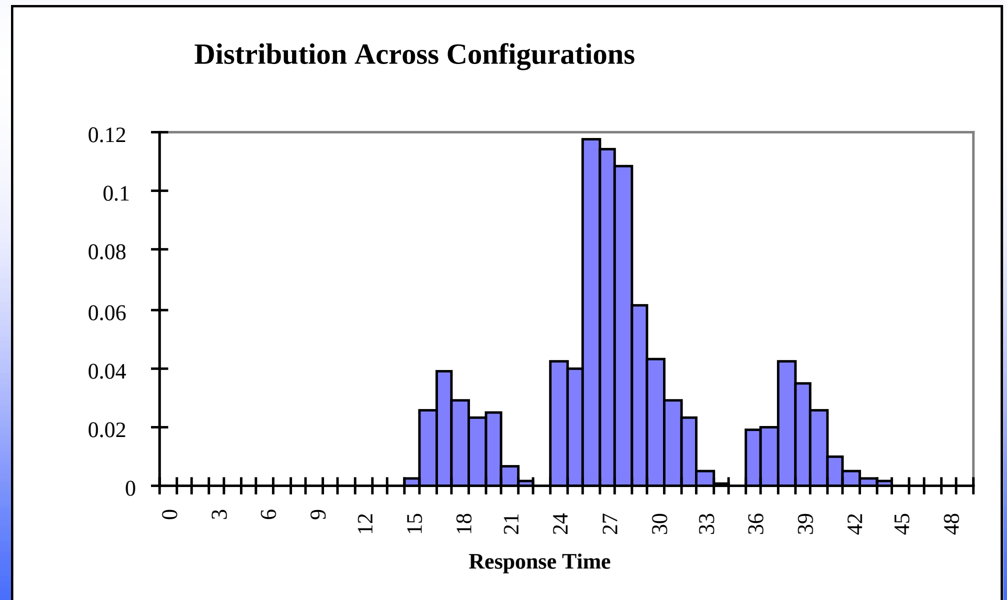




Response Distribution for an Organization



For an organization, response time is combined distribution given *f* (for probabilities of configurations)



Performance Measures: Reliability

- Response time data only for when organization responds
  - minimized when no redundancy!
- Need to factor in reliability
  - penalize organization for brittleness

Performance Measures: Combined RT and Reliability

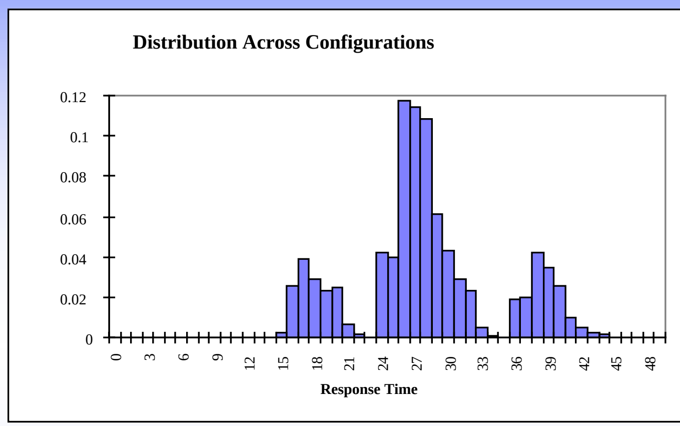
- Give problem to organization
- Restart if no result by max response time (assume random configuration)
- Repeat until succeeds

Probability of success on i^{th} iteration is

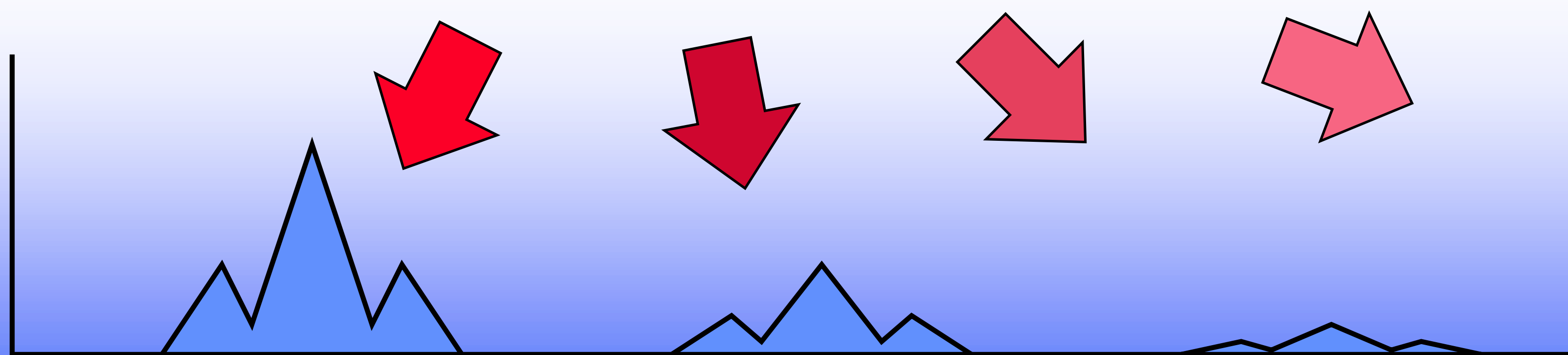
$$f^{i-1} \times (1-f)$$

f is probability of failure on an iteration

Summary Performance Profile

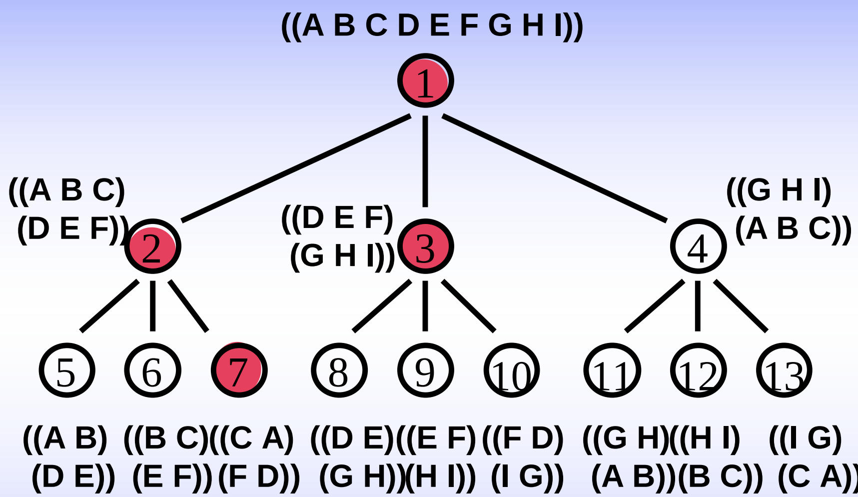


Overall performance is the expected response time of this profile

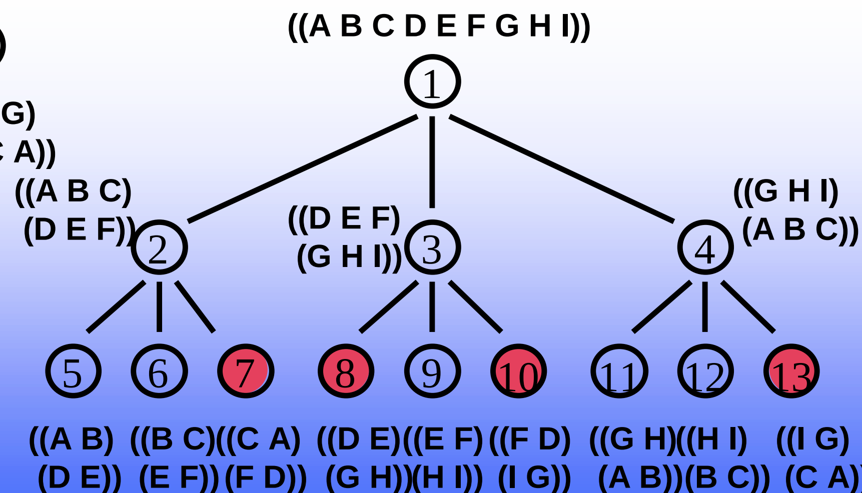


Total area sums to 1

Runtime Coordination Strategies: Role/Task Assignment

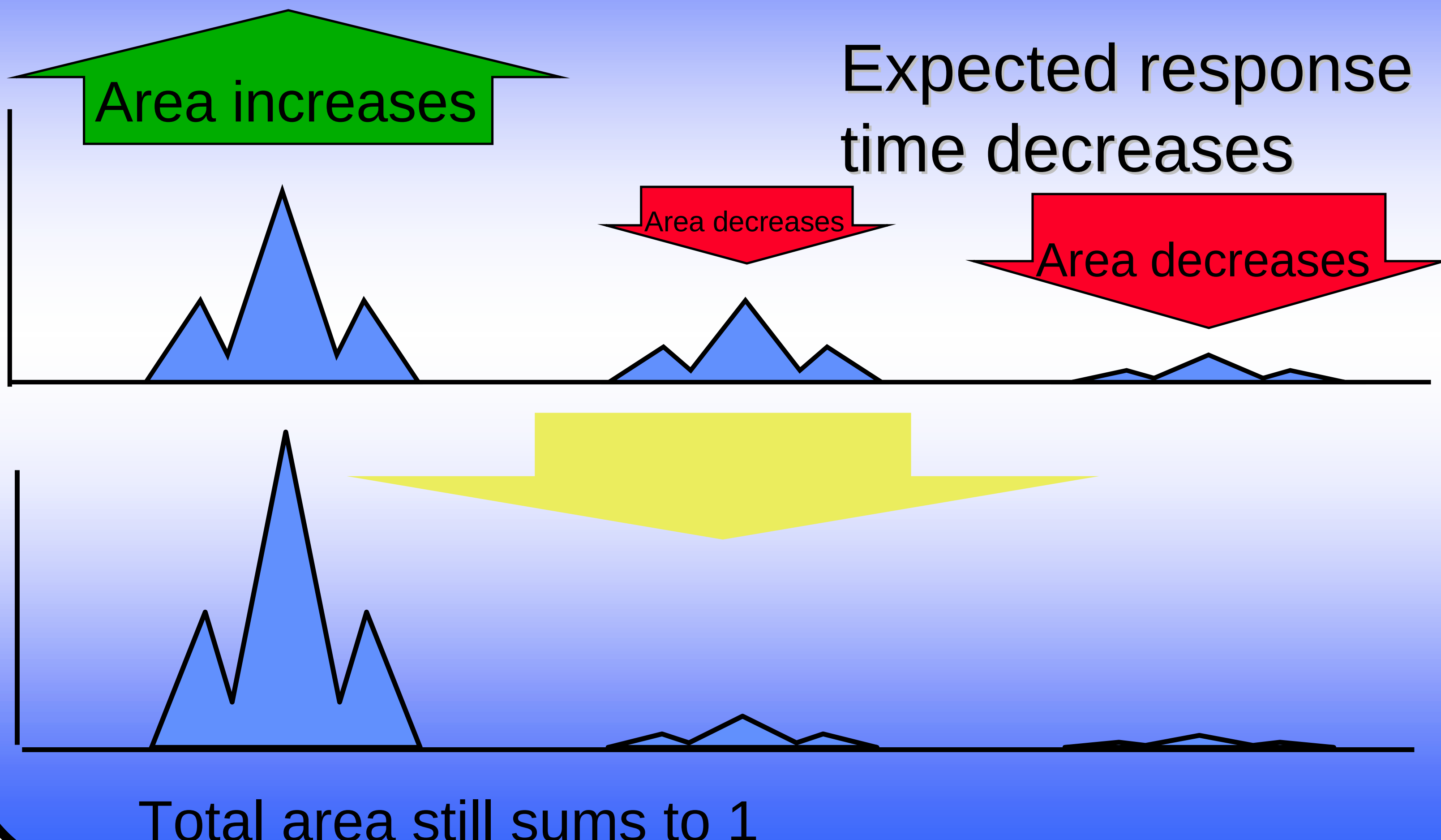


...by (re)allocating roles (same as tasks) at runtime

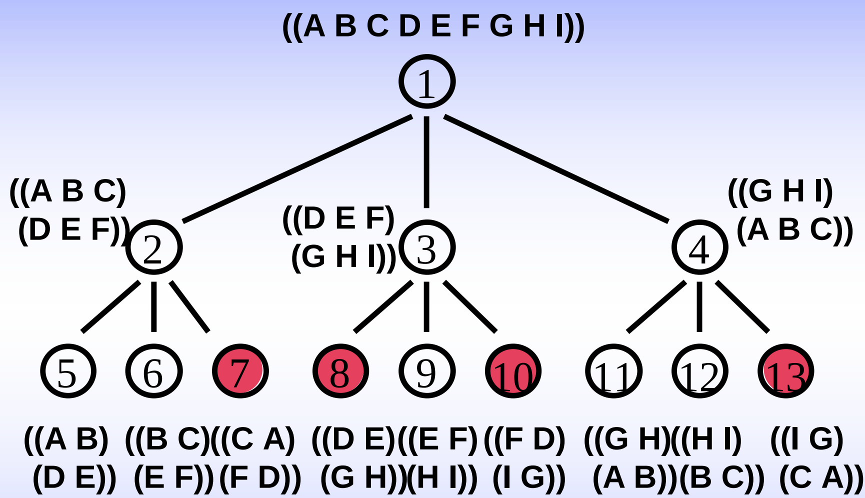


Increase the chances of a working configuration...

Effects of Role/Task (Re)Allocation

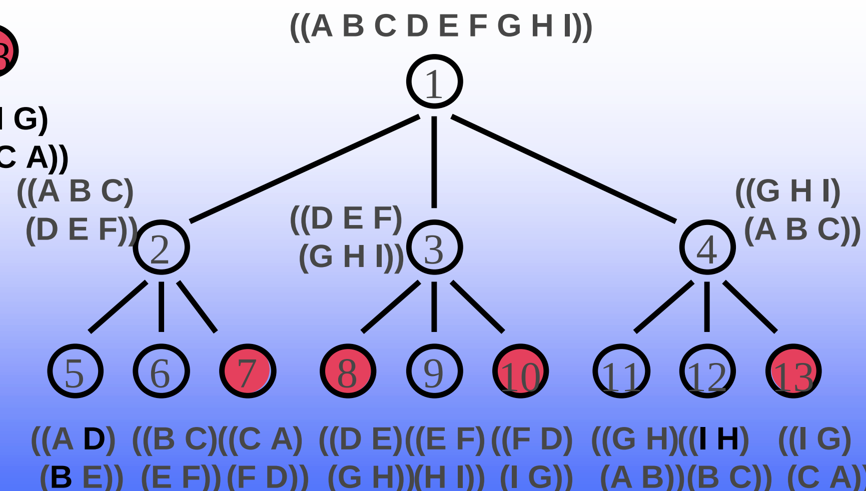


Runtime Coordination Strategies: Task Coordination/Scheduling



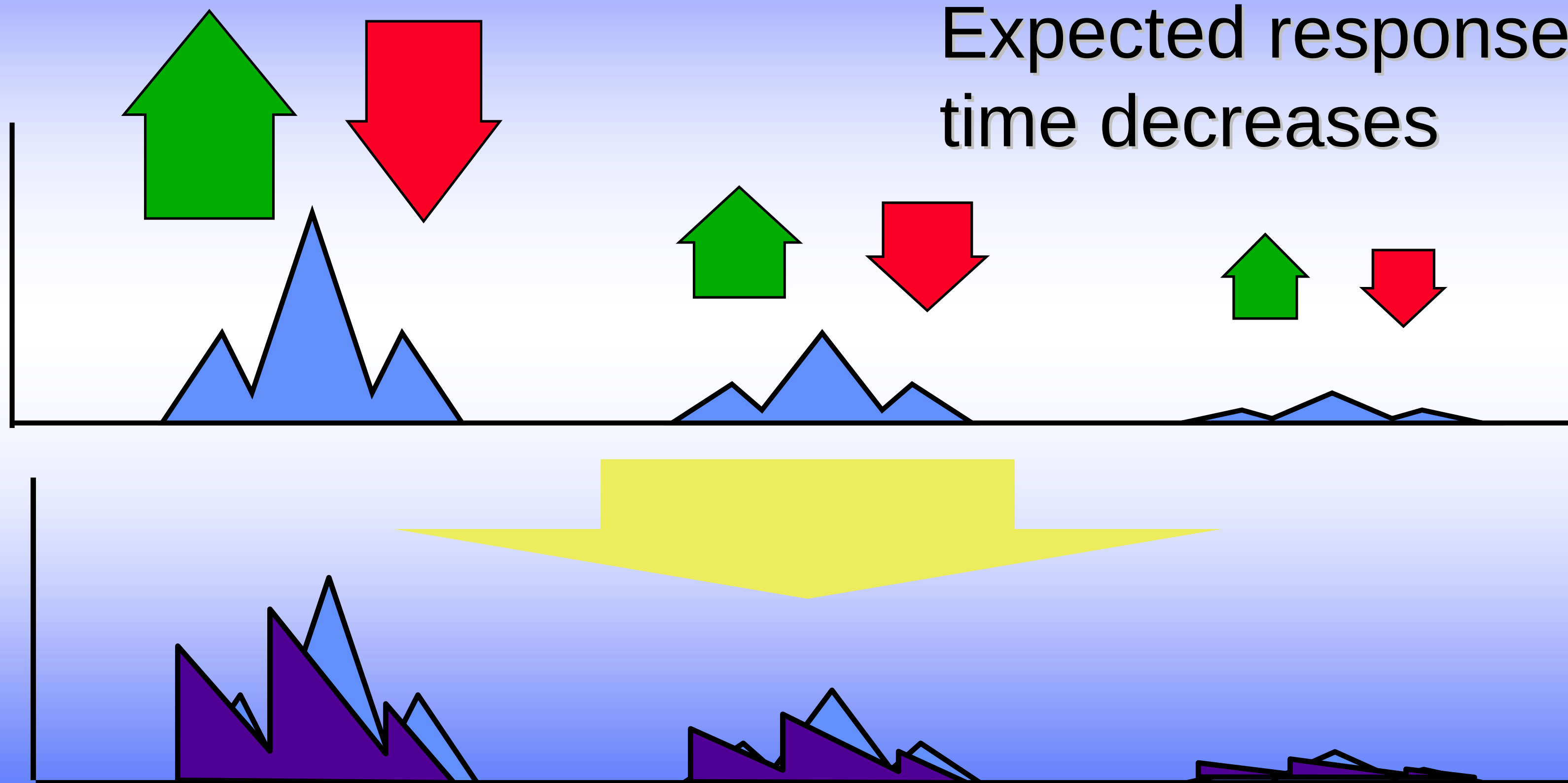
...by (re)ordering tasks at runtime

Decrease the runtime of a working configuration...



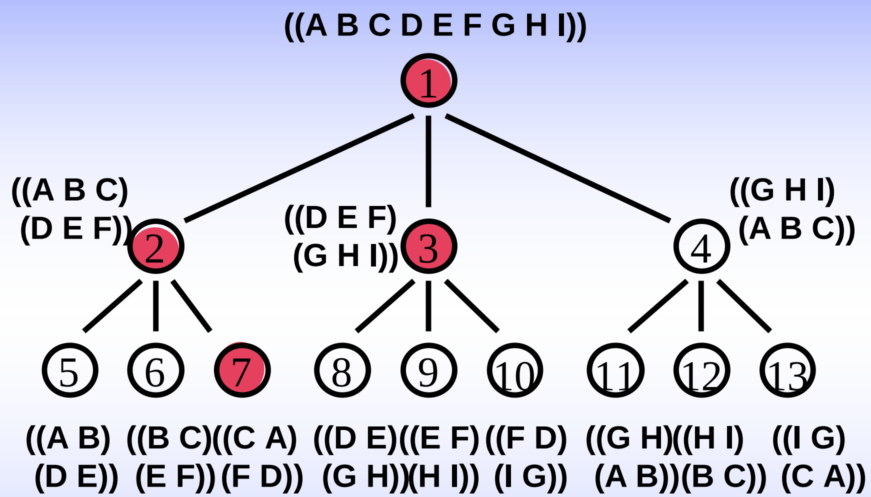
Effects of Local Task Reordering

Expected response time decreases

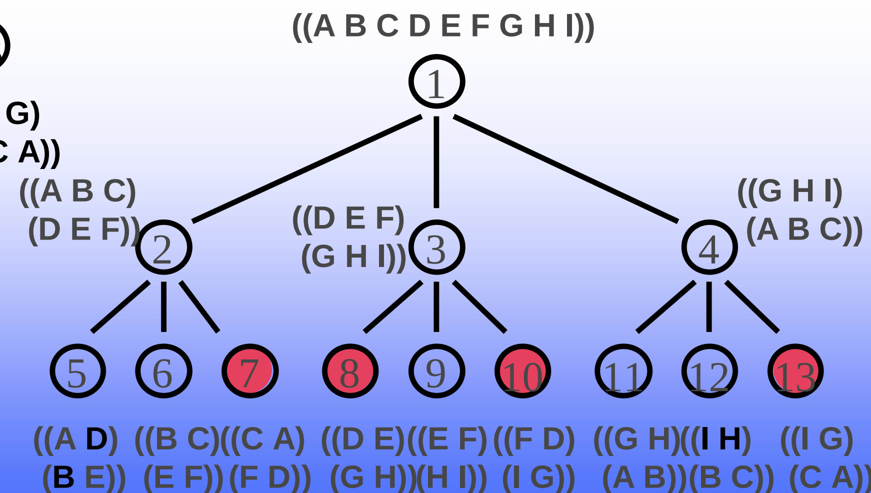


Area under each curve constant

Runtime Coordination Strategies: Combined



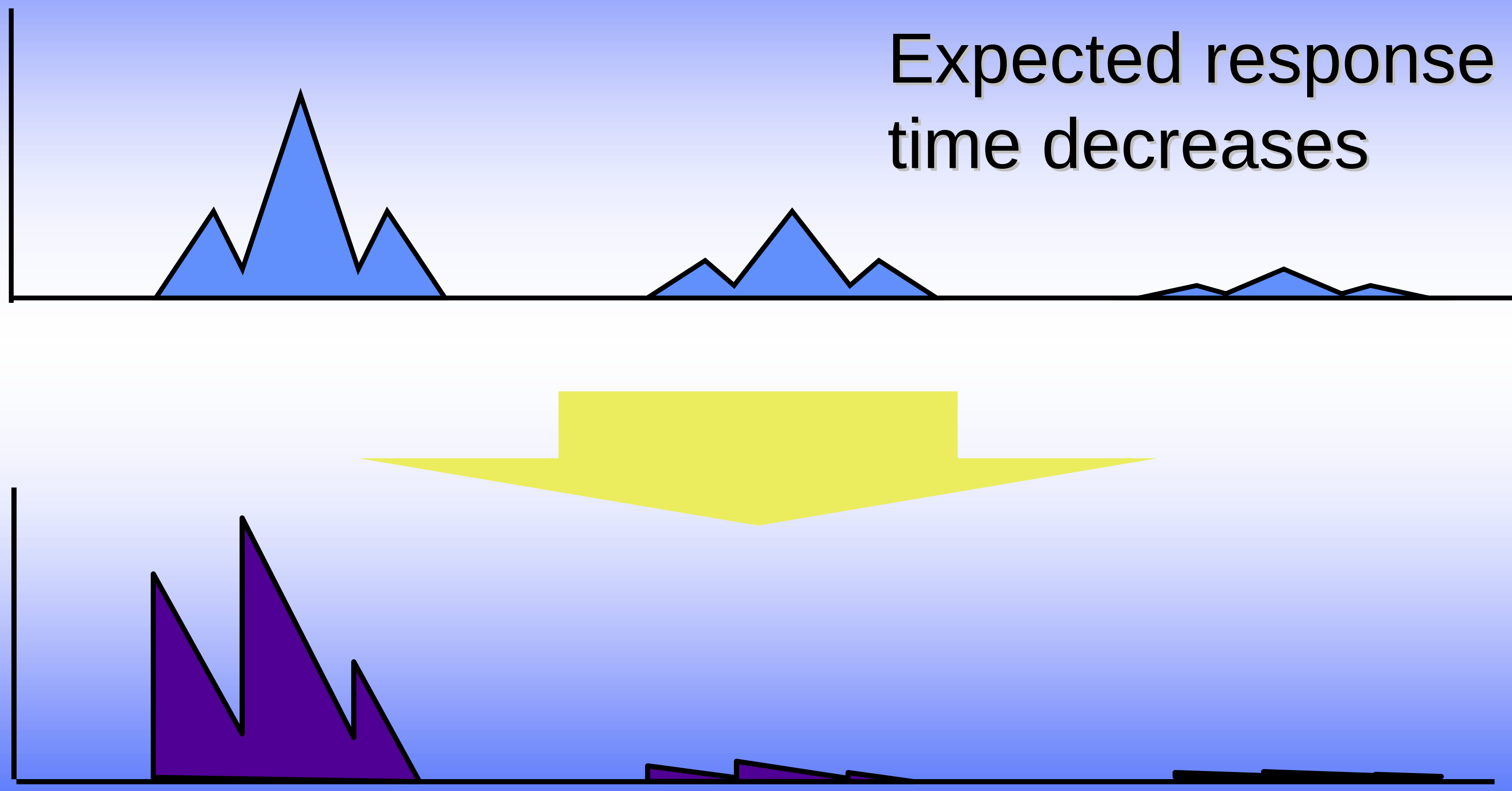
...by (re)allocating roles and (re)ordering tasks



Decrease runtime and number of tries...

Effects of Combined Strategies

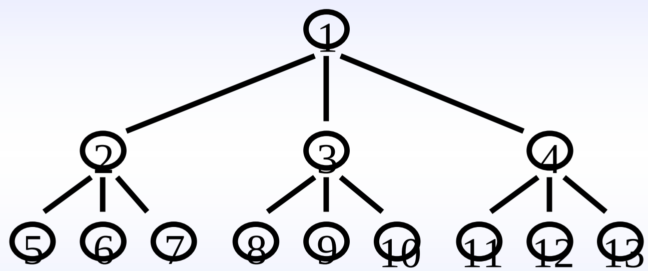
Expected response time decreases



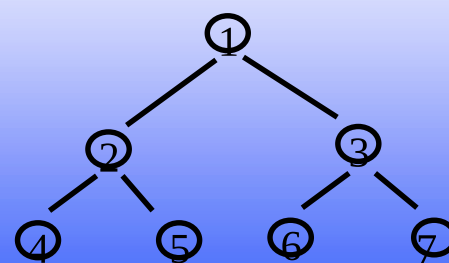
The University of Michigan

Experimental Comparisons

- Number of tasks *N=36*
- Organization 1: branching factor 3, 4 leaf tasks -> 13 roles



- Organization 2: branching factor 2, 9 leaf tasks -> 7 roles



Organizational Structure Comparison

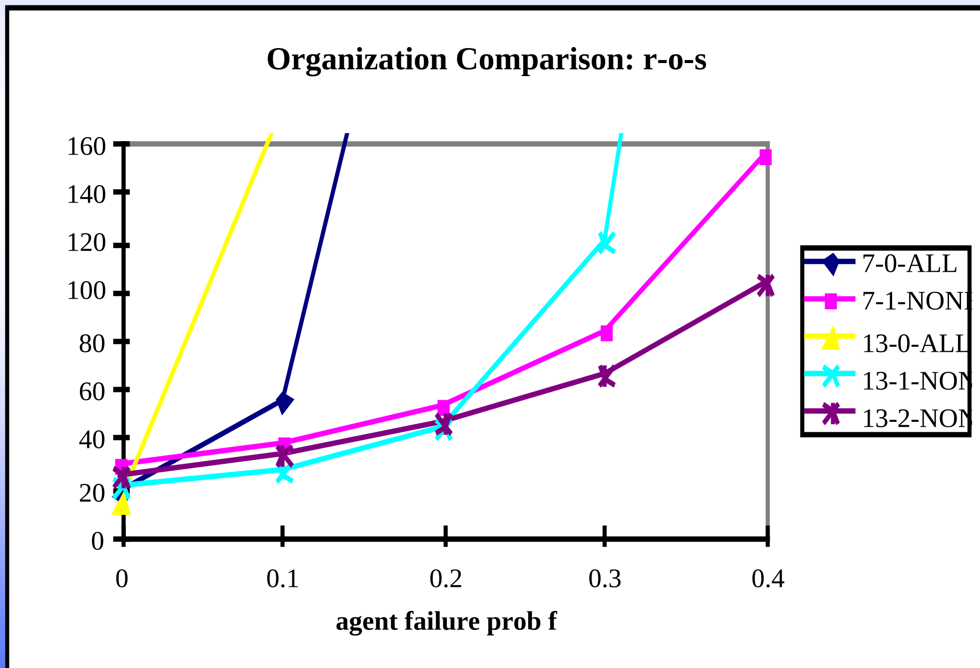
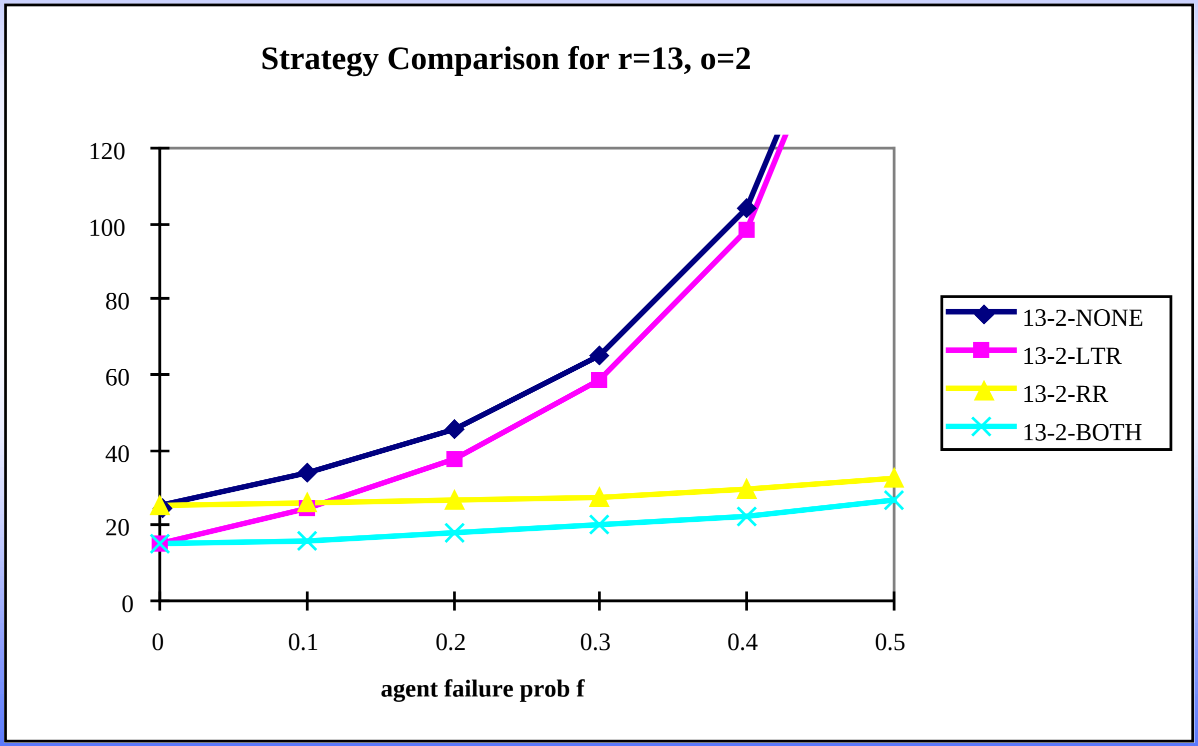


Figure 6

Small org more robust
Large org faster
Redundancy crucial at high failure rates
For same redundancy, (*o* =1) larger org better at low failures, smaller better at high

Runtime Coordination Strategy Comparison



Low failure:
- Local Task Reordering dominates

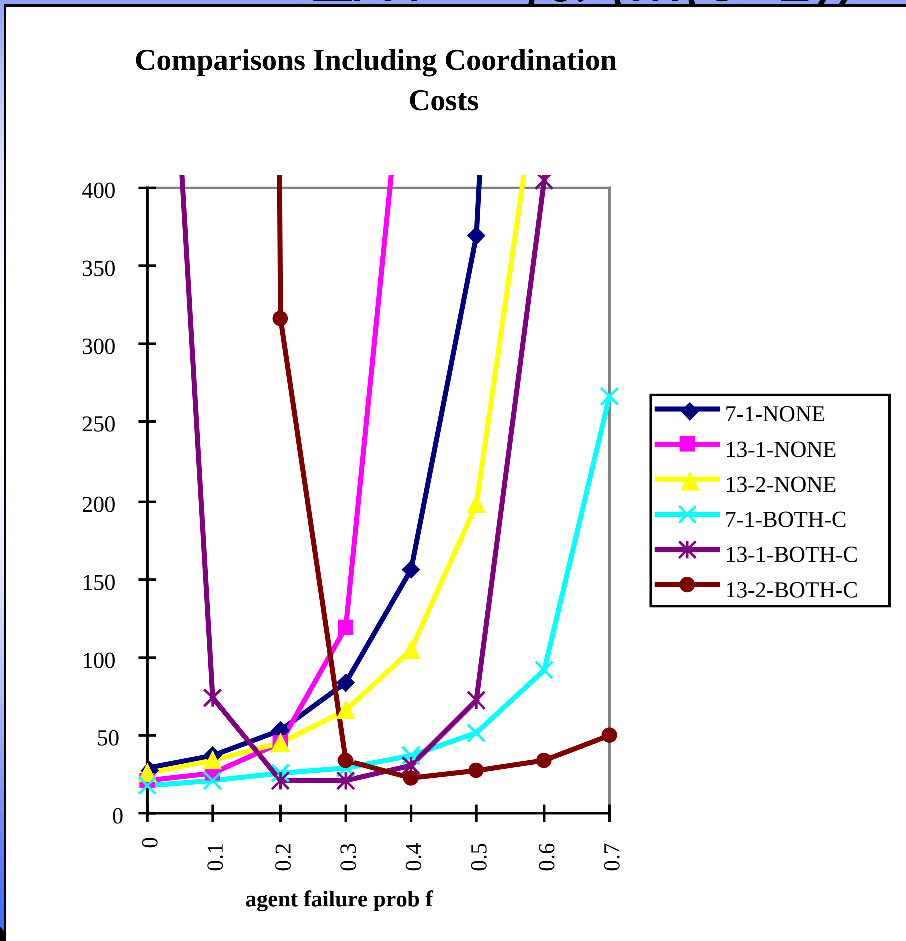
High failure:
- Role Realloc dominates

Organizational Features and Runtime Coordination

- When *o* =0 (no redundancy) neither strategy helps
- As *o* grows
  - LTR matters more, due to more bad orders
  - RR matters more, due to configurations that tolerate even more failures

Factoring in Coordination Costs

$$\Delta RT = [\alpha * (m(o+1))^a + \beta * a^2] * RT$$



Coordination costs rise as number of live agents and number of possible redundancies rise

Choice of org and coord strategies based on expected operating conditions

Task Structures

An analysis about the relationships among the distributed tasks permits identification of agent relationships

Which (partial) results an agent shares, with whom, and when, can enable (or disable) another agent in accomplishing its tasks, or can facilitate or hinder its performance, along with affecting timeliness, confidence, and precision

Other "Sharing" Strategies

Sharing of knowledge or expertise (capabilities)

- Instead of moving tasks to agents that know how to do them, move "know how" to agents that have tasks
- Essentially allows "replication" of agent capabilities in the MAS
- Depending on the nature of the application, moving capabilities might be more efficient (especially for prolonged/repeated tasks) than moving tasks

Planning

How does planning differ from problem solving?

- Russell and Norvig: Planning combines PS and logic – assumes a logical representation of states, goals, and operators
- State is really of knowledge, and thus can be partial, representing multiple “real” states
- Operators manipulate logical expressions, and thus can also remain agnostic about specific worlds (recall Kripke structures?)

Distributed Planning

- In a sense, it is a subset of DPS
  - Makes stronger assumptions about the representations of problem states and operators
  - Assumes that the purpose of problem solving is to construct and/or execute a plan
- Typically goes on concurrently with DPS
  - DPS requires agents to work together in a coordinated manner
  - Coordinating their current and future activities can be done, among other ways, by planning

Kinds of Distributed Planning

- Goal is to formulate a plan, but capabilities or expertise to do so is distributed: DPS with heterogeneous agents where the problem is to construct a plan
- Goal is to have a distributed plan, where each agent has its piece of the plan that, in concert with others, achieves goal
  - Could be formed in a centralized manner
  - Could be formed in a distributed manner

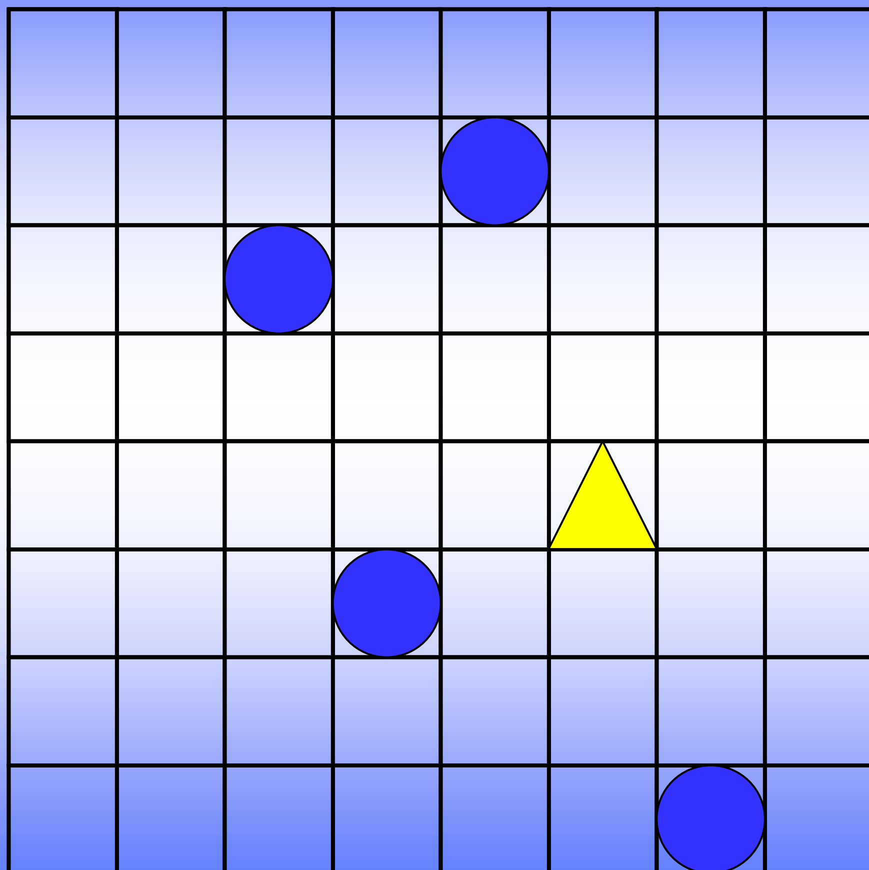
Cooperative Plan Construction

- Employ DPS techniques: allocate portions of plan construction task to “experts”, discover and reconcile constraint conflicts, share and extend partial plans
- Recall cooperative maze solving
- Requires a plan representation and ontology that is understood across multiple agents
- Requires implications of some design decisions to be understood by other agents

Cooperative Plan Construction Examples

- Manufacturing: general fabrication planner calls on specialists in geometry, fixturing
- Logistics planning: overall mission requires contributions from specialists in path planning, vehicle loading and dispatching
- End-to-end communications: experts in routing messages through regions cooperate to form an end-to-end plan

Example 3: Pursuit Task



Example 3: Discussion

Deciding on an assignment of goals to agents

Finding non-conflicting plans for agents to achieve their goals

Timing the pursuit of those plans to converge at a solution at the right time

Assuming global awareness and control, a centralized planner can be very effective!

Centralized Planning for Distributed Plans

For known agents, search for a sequence of simultaneous operator executions (including no-op) that lead to a goal state:

- Model state as the global state
- Branching factor is the number of combinations of applicable local operators

Centralized Planning for Distributed Plans

For exploiting available (homogeneous) agents:

1. Generate a partial order plan with minimal ordering constraints
2. Assign strongly ordered threads to different agents
3. Insert synchronization actions to maintain ordering between agents
4. Allocate partial plans to agents using task-passing
5. Initiate execution

Example 4a: “Blocks World”

Either vertically or horizontally:

Arrange the letters to have “B” and “S” next to each other

Example 4a: “Blocks World”

Either vertically or horizontally:

Arrange the letters to have “O” and “K” next to each other

Example 4a: Discussion

- When agents are pursuing goals that can be achieved via independent plans, having each agent plan separately and not worry at all about the other(s) suffices
- How would the agents have known?
  - Compare plans?
  - Compare goals?
  - Execute and deal with conflicts if and when they arise?

Example 4b: "Blocks World"

Either vertically or horizontally:

Arrange the letters to have the sequence "LOCK" appear

Example 4b: “Blocks World”

Either vertically or horizontally:

Arrange the letters to have the sequence “SOB” appear

Example 4b: Discussion

In some sense, plans/goals were independent, but shared a (sharable) resource

How could you have quickly determined how to coordinate?

- Compared plans?
- Compared specific goal state options?

Plan Merging

Given the candidate plans of the agents, consider all possible combinations of plans, executed in all possible orderings (interleavings or even simultaneous)

Generate all possible reachable sequences of states

For any illegal (inconsistent or otherwise failure) states, insert constraints on which actions are taken or when to ensure that the actual execution cannot fail

Plan Merging Algorithm-1

Each action has pre-conditions, post-conditions, and during-conditions (optional)

1. Compare an agent's actions against each action of the other agents ($O(n^2a)$ comparisons) to detect contradictions between pre, post, and during conditions
2. If none, pair of actions commute and can be carried out in any order.
3. If some, determine if either can precede the other (post-conditions of one compatible with pre-conditions of other)
4. All simultaneous or ordered executions not safe are deemed "unsafe"

Plan Merging Algorithm-2

Ignore actions that commute with all others

Complete safety analysis by propagation

1. Beginning actions a and b is unsafe if either consequent situation (adding post-conds of a to b, or b to a) leads to an unsafe ordering
2. Beginning a and ending b is unsafe if ending a and ending b is unsafe
3. Ending a and ending b is unsafe if both of the successor situations are unsafe

Plan Merging Algorithm-3

In planning, assumption is that plan step interactions are exception

Therefore, dropping commuting actions leaves very few remaining actions

Examining possible orderings and inserting synchronization actions (messages or clock-times) therefore becomes tractable

Example 4c: “Blocks World”

Either vertically or horizontally:

Arrange the letters to have the sequence “SLOB” appear

Example 4c: “Blocks World”

Either vertically or horizontally:

Arrange the letters to have the sequence “LOCK” appear

Example 4c: Discussion

In this case, plans most certainly are not independent.

In fact, goals are conflicting given constraints and resources!

How would you have discovered this efficiently?

- Compared goal states?
- Compared plans?
- Compared constraints?

Example 4d: “Blocks World”

Horizontally:

Arrange the letters to have “B” and “C” separated by 1 or more letters, and “C” and “K” separated by 1 or more letters

Example 4d: “Blocks World”

Horizontally:

Arrange the letters to have the sequence “COSK” in the final arrangement

Example 4d: Discussion

- Solvable (in various ways) but definitely interacting
- Order in which plans are carried out can make a difference:
  - Put “COSK” down, and have other work around is easier than other order
  - Relationship to “most constrained first”?
- More abstract representation can come in handy
  - Isolate and focus on the “C” to “K” portion

Iterative Plan Formation

Sometimes, forming plans first and then coordinating them fails because of choices in initial plans formed

Instead, iterate between formation and coordination to keep alternatives alive

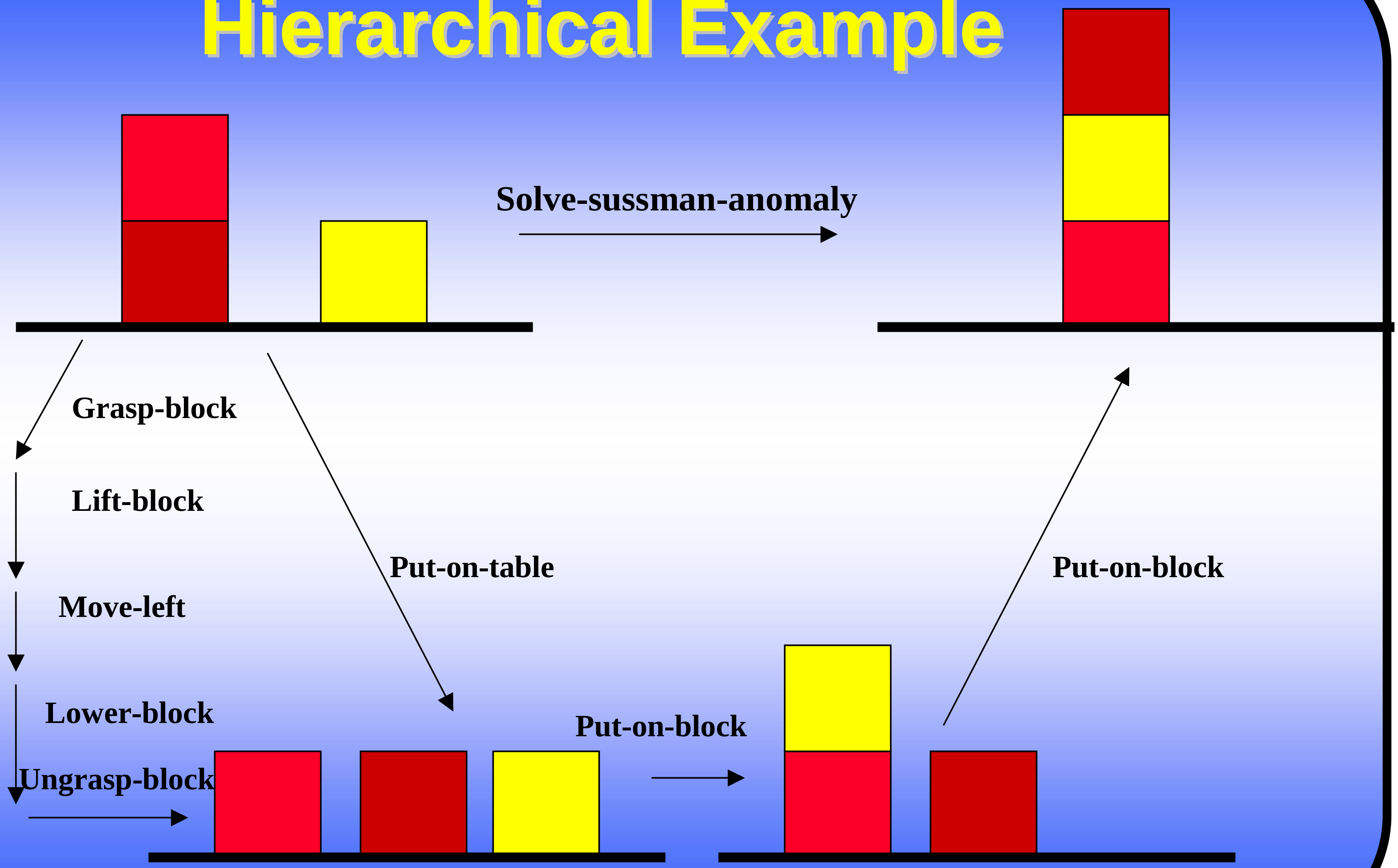
Plan Combination Search

Given initial propositions about the world

1. Agents form successor states by proposing changes to current propositions caused by one action (or no-op)
2. Successor states are ranked using A* heuristic by all agents, and best choice is found and further expanded

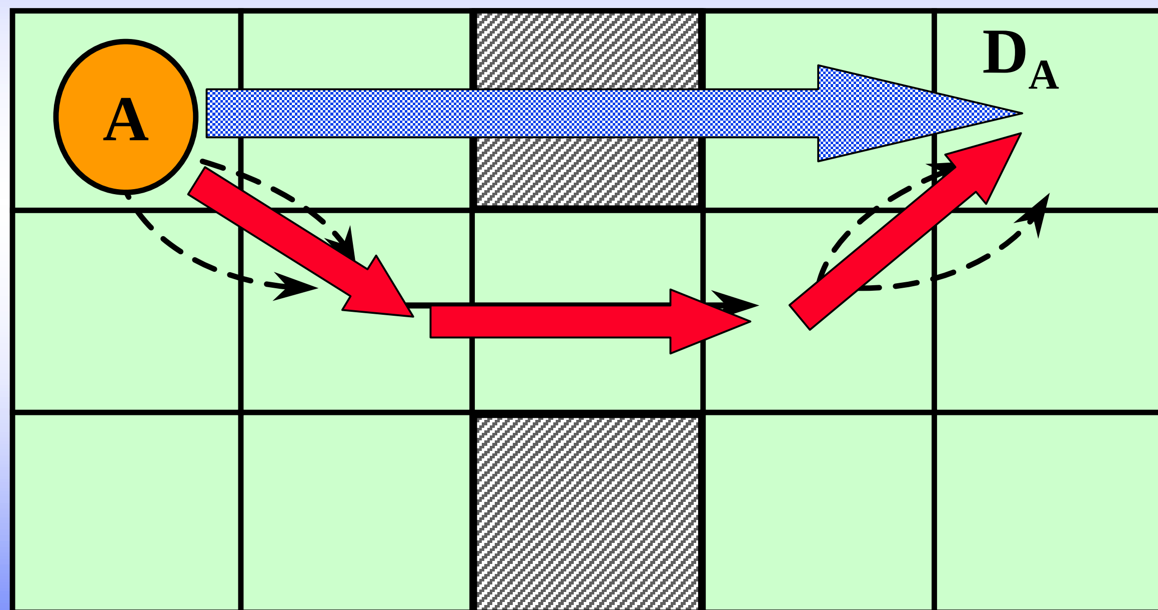
Agents are simultaneously committing to a plan (corresponding to actions in solution path) and synchronizations (when actions are taken relative to each other)

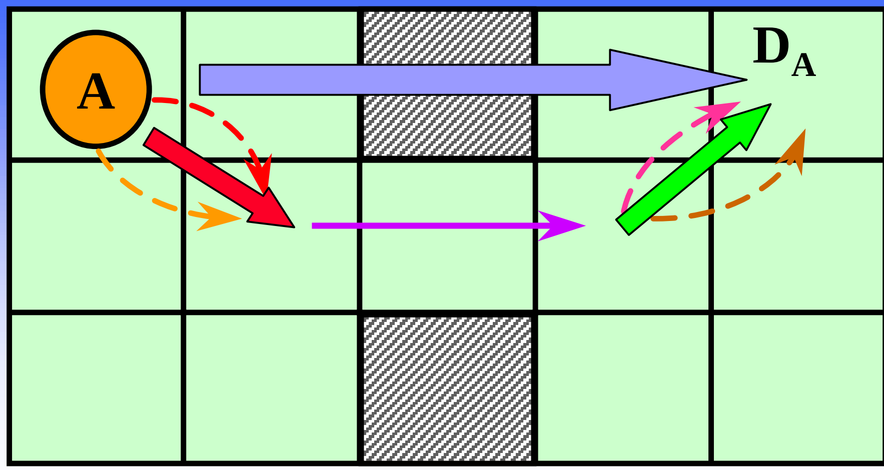
Hierarchical Example



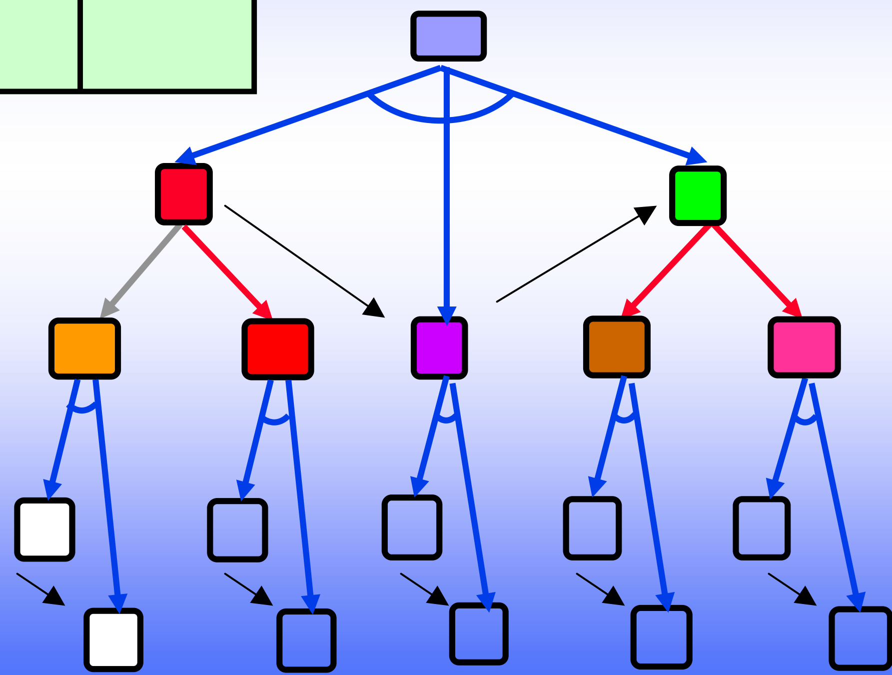
The University of Michigan

Another Hierarchical Example



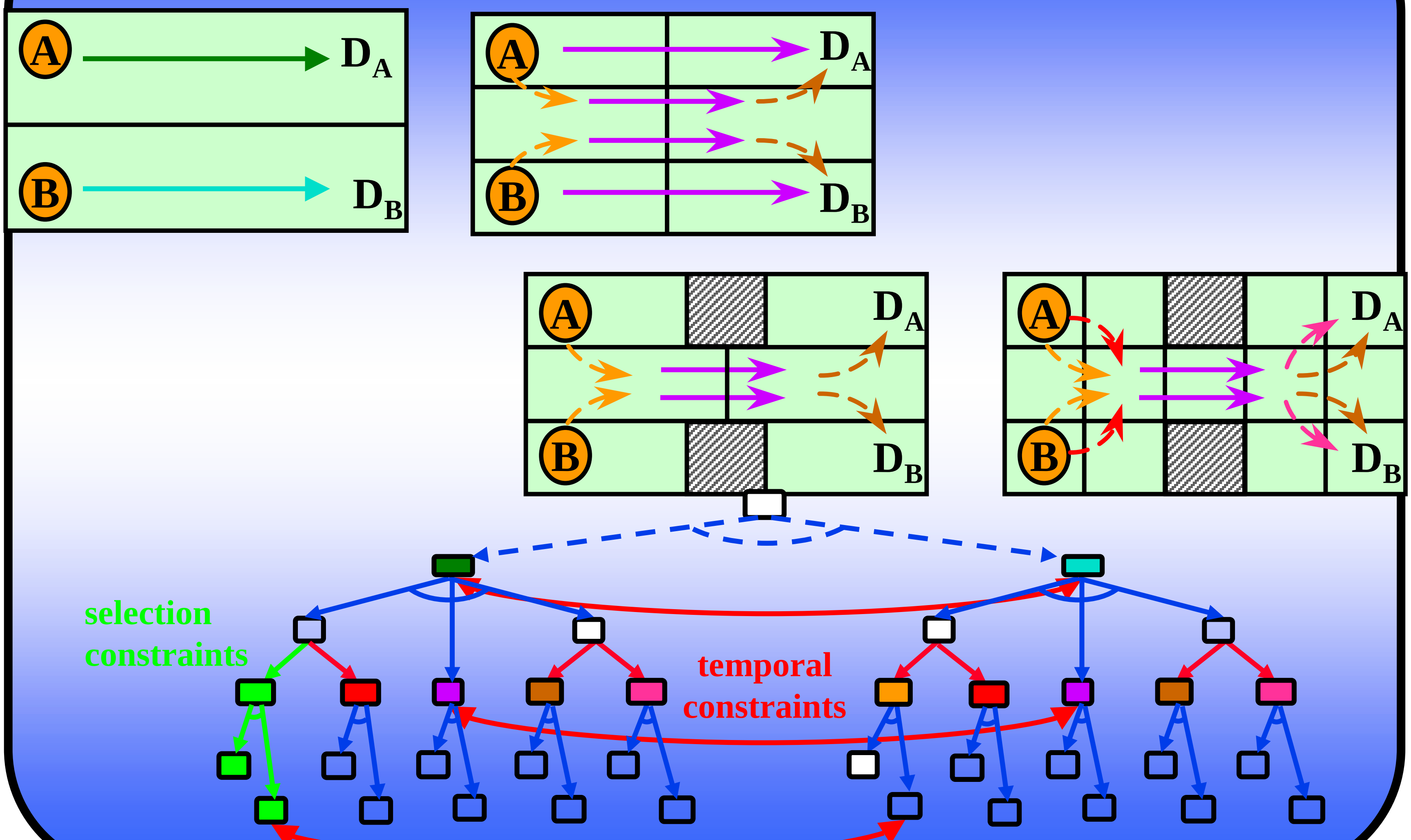


Hierarchical Plan



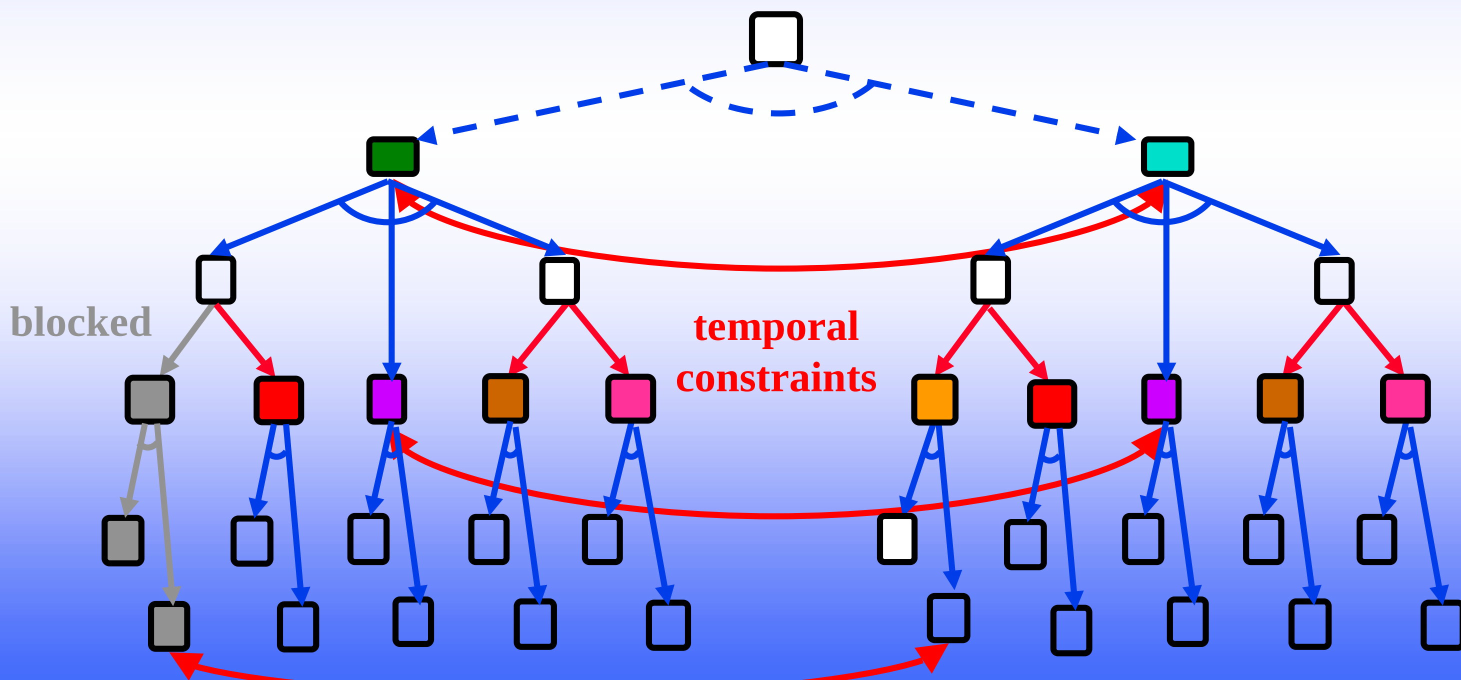
The University of Michigan

Multi-level Coordination & Planning



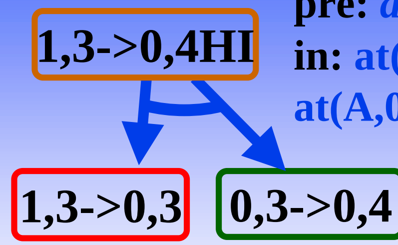
Top-Down Search

- Know as little as you can about others.
- Use abstract resolutions to obviate deeper ones.
- How can you know constraints between abstract levels without having expanded/investigated lower levels?



Summary Information

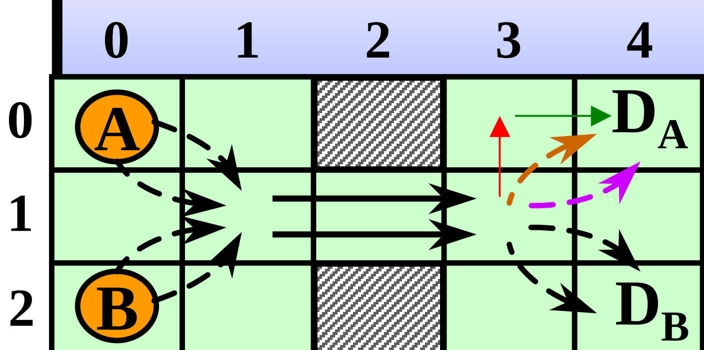
- must, may
- always, sometimes
- *first, last*
- external preconditions
- external postconditions



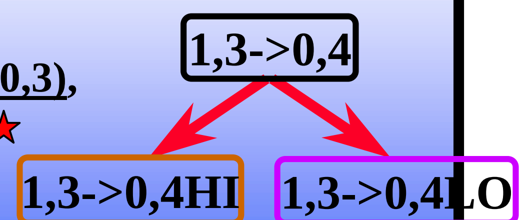
pre: *at(A,1,3)* ★
in: at(A,1,3), ¬at(B,1,3), ¬at(B,0,3) ★, at(A,0,3), ¬at(B,0,4), ¬at(A,1,3)
post: ¬at(A,1,3), ¬at(B,1,3), *¬at(B,0,3)*, *at(A,0,4)*, *¬at(A,0,3)*, *¬at(B,0,3)*, *¬at(B,0,4)*

pre: *at(A,1,3)*
in: at(A,1,3), ¬at(B,1,3), ¬at(B,0,3) ★
post: *at(A,0,3)*, *¬at(A,1,3)*, *¬at(B,1,3)*, *¬at(B,0,3)*

pre: *at(A,0,3)*
in: at(A,0,3), ¬at(B,0,3) ★, ¬at(B,0,4)
post: *at(A,0,4)*, *¬at(A,0,3)*, *¬at(B,0,3)*, *¬at(B,0,4)*

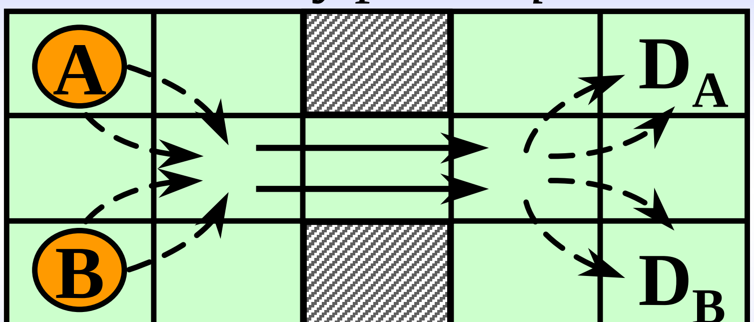


pre: *at(A,1,3)*
in: at(A,1,3), ¬at(B,1,3), ¬at(B,0,3), ¬at(B,1,4), at(A,0,3), at(A,1,4) ★, ¬at(B,0,4), ¬at(A,1,3)
post: ¬at(A,1,3), ¬at(B,1,3), *¬at(B,0,3)*, *at(A,0,4)*, *¬at(A,0,3)*, *¬at(A,1,4)*, ¬at(B,0,3), ¬at(B,1,4), *¬at(B,0,4)*



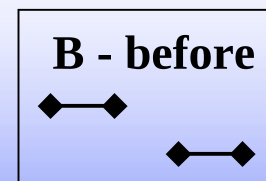
Determining Temporal Relations

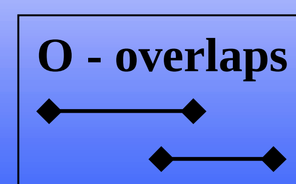
- CanAnyWay(*relation*, p_{sum}, q_{sum}) - *relation* **can** hold for **any way** *p* and *q* can be executed
- MightSomeWay(*relation*, p_{sum}, q_{sum}) - *relation* **might** hold for **some way** *p* and *q* can be executed



CanAnyWay(before, p_{sum}, q_{sum})
¬CanAnyWay(overlaps, p_{sum}, q_{sum})
MightSomeWay(overlaps, p_{sum}, q_{sum})

- CAW used to identify solutions
- ¬MSW used to identify failure
- CAW and ¬MSW improve search
- ¬CAW and MSW → must look deeper
- MSW identifies threats to resolve

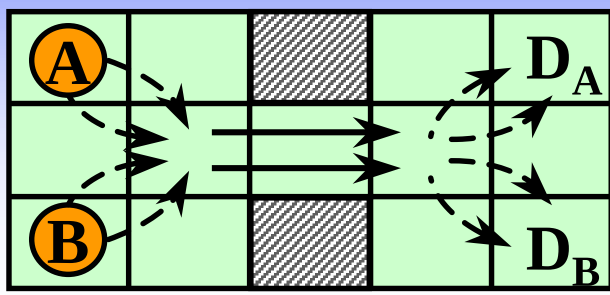




Hierarchical Coordination Search

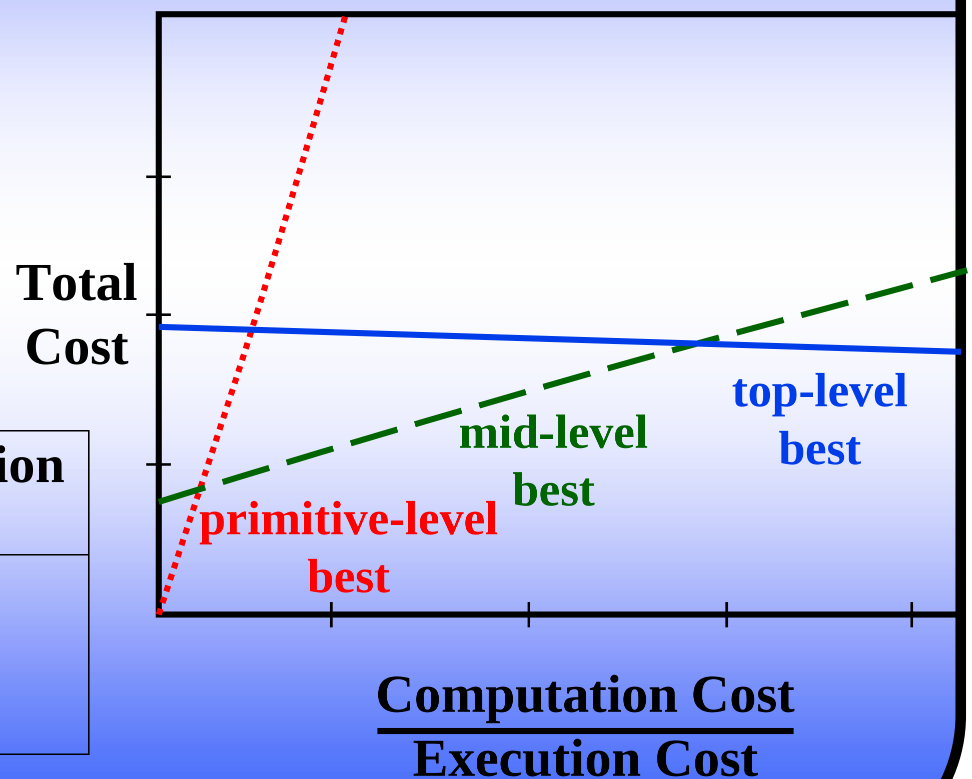
1. Initialize the current abstraction level to most abstract
2. Agents exchange descriptions of their plans and goals at the current level
3. Remove plans or plan steps with no potential conflicts. If nothing left, done. If conflicts should be resolved at this level, skip next step.
4. Set the current level to the next deeper level, and refine all remaining plans (steps). Goto 2.
5. Resolve by: (i) put agents in a total order; (ii) current top agent sends its plans to others; (iii) lower agents change plans to avoid conflicts with received plans; (iv) next lower agent becomes top agent

Coordinating at Abstract Levels Can Improve Performance



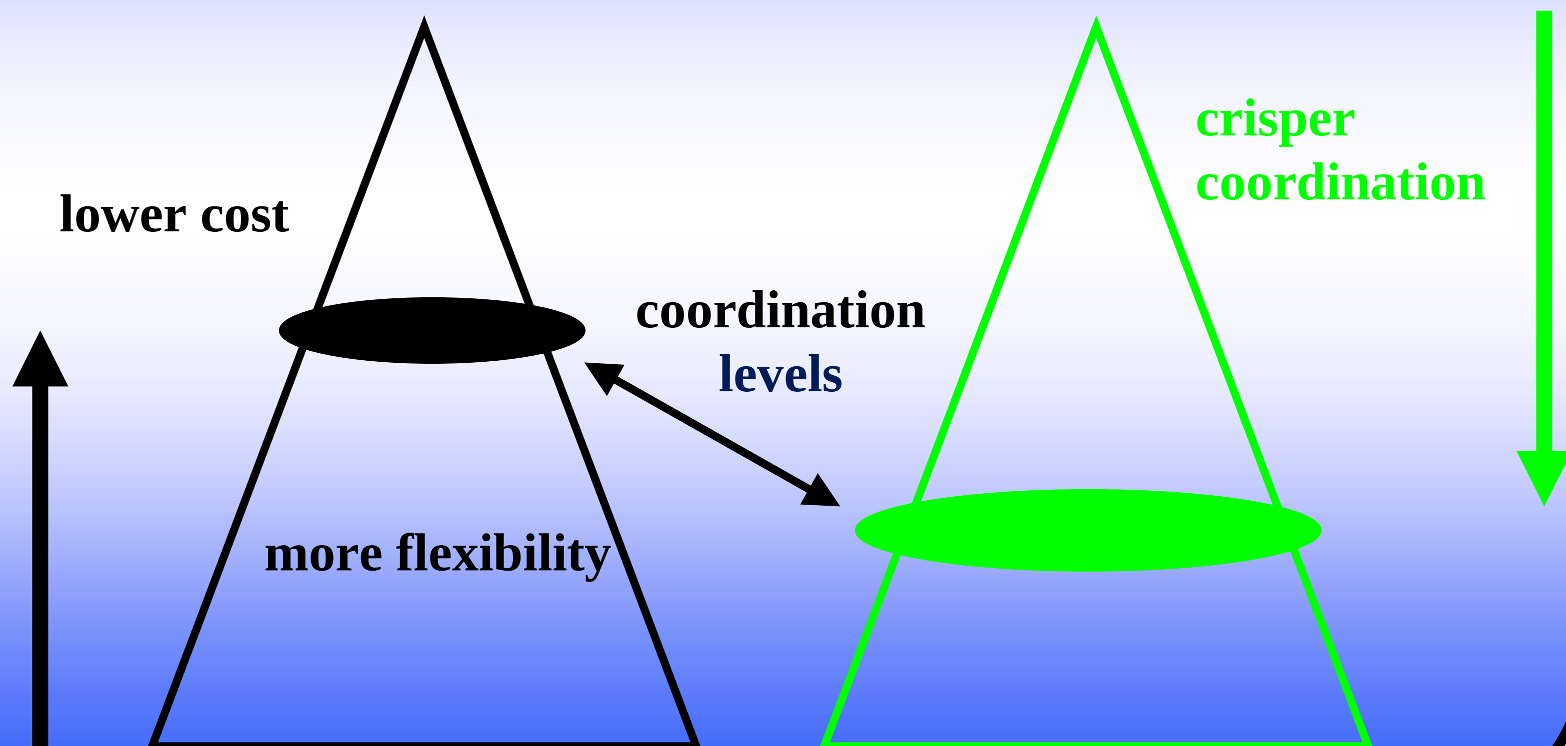
BFS algorithm

level	computation time	execution time
top	4	60
mid	159	40
primitive	2375	35



Tradeoffs

Choice of level at which coordination commitments are made matters!



Example 4e: “Blocks World”

Horizontally:

Arrange the letters to have the letter “S” surrounded by other letters

Example 4e: “Blocks World”

Horizontally:

Arrange the letters to separate “C” and “O” by exactly 1 letter

Example 4e: Discussion

Sometimes, one agent's actions, if chosen properly, can help another satisfy its goals

- Coordinating plans not just to avoid conflicts
- Synergistic interactions such that the total effort for coordinated plans less than the sum of the efforts of stand-alone plans
- Issue is how much extra effort goes into finding the synergies, and is it less than what is ultimately saved?

Distributed Planning and Execution

Issues in when agents plan and coordinate, relative to each other, and relative to execution

Are often sequentialized

No sequential order works well in all cases

Post-Planning Coordination

- Essentially, plan merging techniques
- Dealing with execution problems can involve:
  - Contingency preplanning: detecting multiagent contingency, and invoking already coordinated response
  - Monitoring/replanning: detecting deviation and restarting the planning/coordination process
- Obviously, localizing impacts minimizes fresh coordination; building a plan that permits localized adjustments can be important, but might be less efficient

Pre-Planning Coordination

Impose coordination constraints before planning is done; plans work within these

Example: Set the boundaries; define the roles

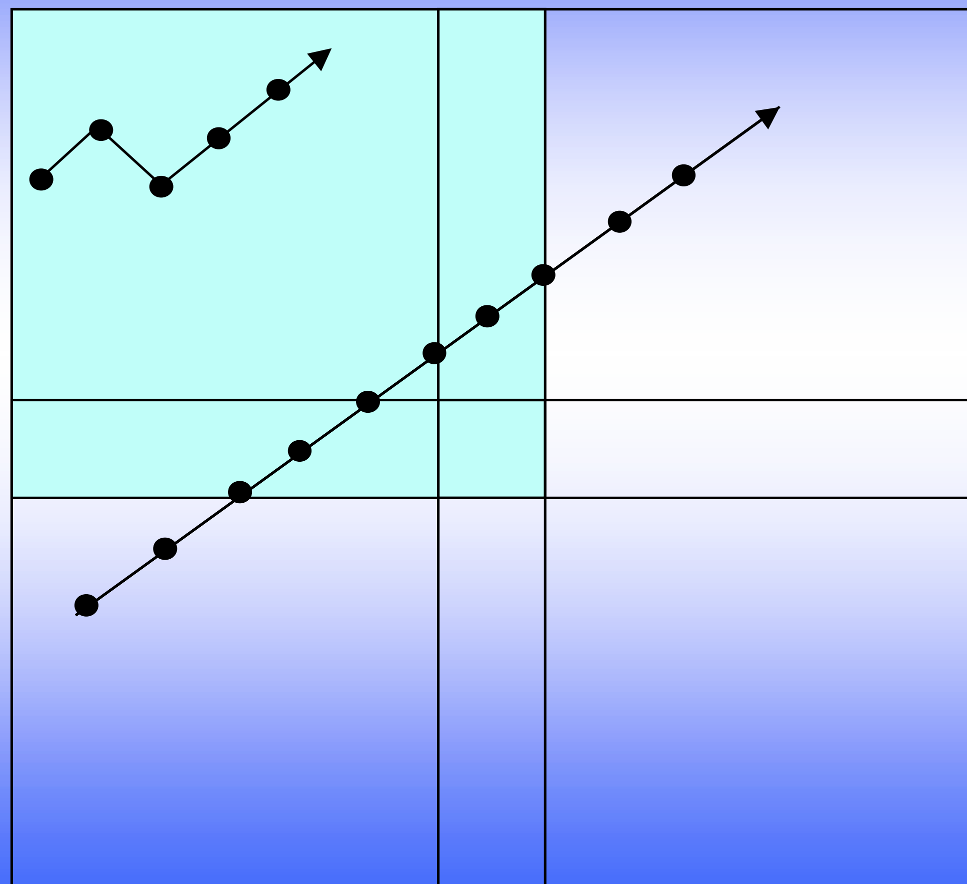
Social laws: Define what could be done and when, then leave it up to agents to plan within the legal limits

Cooperative state changing rules: Force agents planning decisions into cooperative behaviors

Continual Distributed Planning and Execution

- Planning, coordination, and execution are all asynchronously interleaved
- At any given time, plans might only be partially coordinated, and execution results could cause chain reactions of further planning and coordination
- In a sense, the coordinated plans are only evident after the fact, as they are continually being adjusted during execution

Example Application: Distributed Vehicle Monitoring



Partial Global Planning

1. Task allocation: inherent
2. Local plan formulation: sequence of interpretation problem solving activities
3. Local plan abstraction: major plan steps (such as for time-region processing)
4. Communication: Use meta-level organization to know who is responsible for what aspects of plan coordination

Partial Global Planning (cont)

5. Partial global plan construction: Pieces of related plans (e.g., potentially tracking the same vehicle) are aggregated
6. Partial global plan modification: redundant or inefficient schedules are adjusted to improve collaborative performance
7. Communication planning: identification of partial results that should be gainfully exchanged, and when

Partial Global Planning (cont)

8. Mapping back to local plans: Partial global plan commitments are internalized
9. Local plan execution

Cycle repeats as local plans change or new plans from other agents arrive. Always acting on local information means that there could be inconsistencies in global view, but these are tolerated

Controlling Continual Distributed Planning

- Danger of constant chain reactions of minor changes: more effort expended in making minor adjustments than saved in having better coordinated plans!
- Agent needs to have a threshold for tolerating obsolete plans/coordination
- Better load balancing also by reallocation of tasks (individually) or roles (in organization) or coordination responsibilities (in MLO)

Continual Distributed Planning: Other Ideas

Markov Decision Process Models (Boutilier)

Teamwork models (Tambe; Grosz)

Runtime Coordination without Communication

Observation-based coordination

Focal points

Open Problems

- Chicken-and-egg: decomposition/allocation
- Ontologies
- Knowing when local information (partial results, or local plan changes) will make a sufficiently useful difference to send them
- Knowing when to continue with partial coordination, and when to wait for convergence
- Finding synergies, and knowing when to
- Measures of "better" coordinated plans, including aspects of robustness
- Instilling social knowledge and context to internalize coordination inherently