Subgoal Ordering and Granularity Control for Incremental Planning

Chih-Wei Hsu, Benjamin W. Wah, and Yixin Chen
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
URL: http://manip.crhc.uiuc.edu

IEEE International Conference on Tools for AI, 2005

Outline

- Introduction
- Approach
  - Incremental planning by subgoals
- Issues addressed
  - Initial ordering of subgoals
  - Recovery from failed subgoals in basic planners
  - Granularity control
  - Trade-offs between solution quality and search time
- Demonstration of improvements over basic planners
- Conclusions

Motivations

- In planning under uncertainty in dynamic domains, a planner needs to
  - React to new uncertain events
  - Maintain valid prefixes of short-term goals that have been achieved
- Decompose a large planning problem into a sequence of smaller subproblems
  - Much easier to satisfy a subset of requirements
  - Important to maintain high solution quality

Problems Addressed

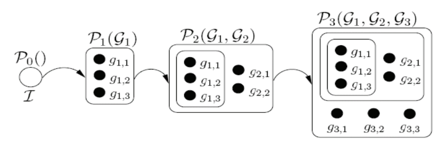
- Solve planning problems in STRIPS representation
  - P = (F, O, I, G):
    - F: set of facts
    - O: set of operators
    - I: set of facts in the initial state
    - G: set of facts in the goal state
  - Generate a sequence of actions from I to G
- Goals
  - Solve more instances than existing (basic) planners
  - Improve run time and quality

Outline

- Introduction
- Approach
  - Incremental planning by subgoals
- Issues addressed
  - Initial ordering of subgoals
  - Recovery from failed subgoals in basic planners
  - Granularity control
  - Trade-offs between solution quality and search time
- Demonstration of improvements over basic planners
- Conclusions

Incremental Planning by Subgoals

- Solve a planning problem in a multi-step fashion
  - Decompose goal facts into several subproblems
  - Achieve in each step all the goal facts in this and previous stages
- Issues
  - Order and group goal facts for best performance
  - Integrate with basic planner



Assumptions

- Achieve a set of goal facts in each stage with
  - Ordering constraints among subgoals
  - Durative actions (in addition to duration of one)
- But without
  - General state trajectory constraints such as deadlines
  - Optimization metric

Outline

- Introduction
- Approach
  - Incremental planning by subgoals
- Issues addressed
  - Initial ordering of subgoals
  - Recovery from failed subgoals in basic planners
  - Granularity control
  - Trade-offs between solution quality and search time
- Demonstration of improvements over basic planners
- Conclusions

Initial Ordering of Subgoals

- Crucial for runtime efficiency and solution quality
  - For an order that causes no subgoal invalidations
    - A small number of new subgoals solved in each step
  - For an order that causes subgoal invalidations
    - Extra actions and search time to re-achieve invalidated subgoals
    - Backtracking to a different order if the order leads to an infeasible plan
- Evaluation metric
  - Number of invalidations

Previous Reasonable-Ordering Algorithm

- Detect partial order between subgoals *i* and *j* (Koehler'00)
  - Order *i* after *j* if any plan that reaches *j* must invalidate *i*
- Resolve loops and unordered pairs to produce a total ordering
  - May group all subgoals involved in unresolved partial-order relations together
- Deficiencies of reasonable ordering
  - Only analyze subgoal interactions without considering initial state
  - Invariant order for any initial state is rare in practice
  - Little ordering information found in IPC-4 domains
    - Some partial orders in 42 out of 50 instances in Airport, 42 out of 100 instances in Pipesworld, and none in the other domains

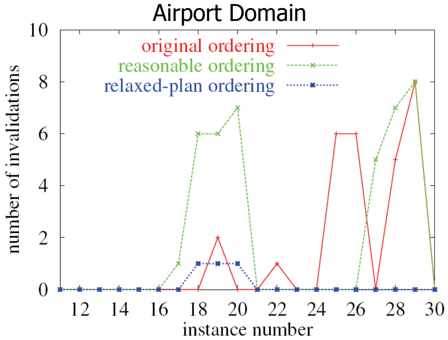
Proposed Relaxed-Plan Ordering

- Use the order of subgoals achieved in relaxed plan
  - Consider the initial state of relaxed plan
  - Use the original order for subgoals at the same level
  - Total ordering
- Little overhead since relaxed plans are widely used for guidance in planning
- Detect a superset of partial-order information than reasonable ordering
  - Relaxed-plan ordering is specific to the initial state and the current planning graph
  - Reasonable ordering is independent of initial state and is stronger
- No constraints on grain size

Relaxed-Plan Ordering Algorithm

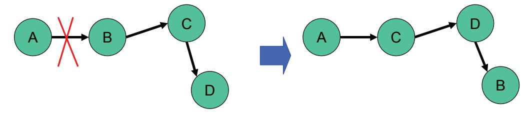
procedure Relaxed-Plan-Ordering $(\mathcal{P} = (\mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G}))$
  construct planning graph G from $\mathcal{I}$ to $\mathcal{G}$ without computing mutual exclusions;
  extract a relaxed plan from G;
  for each pair of subgoals g_i and g_j
    set PREC ⟵ **true**;
    for each action o in P that has g_j as an add effect
      if $(g_i \not\subseteq del(o))$ **and** $(pre(o) \cap F(g_i) == \varnothing)$
      then PREC ⟵ **false**;
    end_for
    if PREC == **true**
    then order g_j before g_i;
    else if g_j is reached before g_i in the relaxed plan
      then order g_j before g_i;
      else order g_i before g_j;
  end_for
end_procedure

Comparison of Three Orders



Dynamic Reordering

- Embedded planner may take too long to complete one stage
- Alternative orders may alleviate this problem
  - Reorder subgoals to achieve easy ones first
    - Estimate difficulty and search-time limit based on the length of the relaxed plan for unsatisfied subgoals
  - Invoke dynamic reordering if the number of states evaluated exceeds a threshold



Granularity Control

- Trade-offs between grain size and total time to solve overall problem

Satellite-15, 24 subgoals

N	1	2	3	4	6	**12**	24
$\|G_i\|$	24	12	8	6	4	**2**	1
T_s	6.6	2.2	1.4	1.1	0.03	**0.002**	0.002
T_{total}	6.6	4.3	4.2	4.3	0.2	**0.02**	0.06

- Initially partition all goal facts into ten equal-sized subsets
- Double the grain size if number of states evaluated is less than a predefined threshold (4)
  - Additive increases may not find a good grain size quickly
  - Insignificant improvements when the number of stages is small

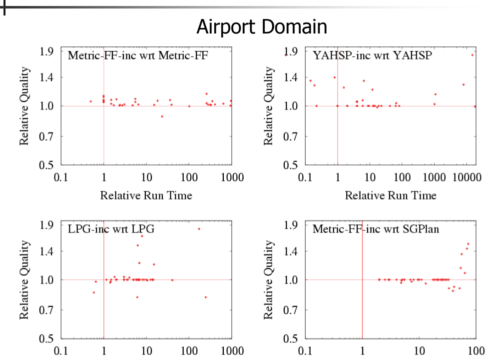
Outline

- Introduction
- Approach
  - Incremental planning by subgoals
- Issues addressed
  - Initial ordering of subgoals
  - Recovery from failed subgoals in basic planners
  - Granularity control
  - Trade-offs between solution quality and search time
- Demonstration of improvements over basic planners
- Conclusions

Experiment Results

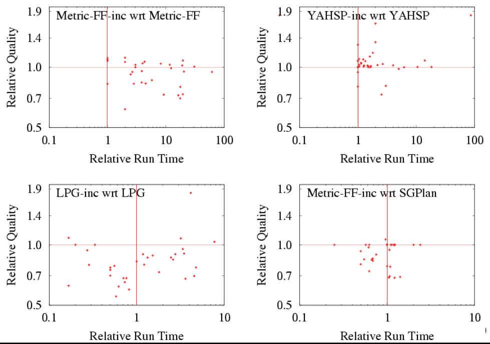
- Evaluation of all IPC4 STRIPS domains and some domains with a lot of goal interactions
- 3 basic planners: Metric-FF, YAHSP, LPG-TD-speed
- Comparison of search time (number of actions) and solution quality
  - Normalize performance measures with respect to those of non-incremental version
  - Also compare with SGPlan

Normalized Performance



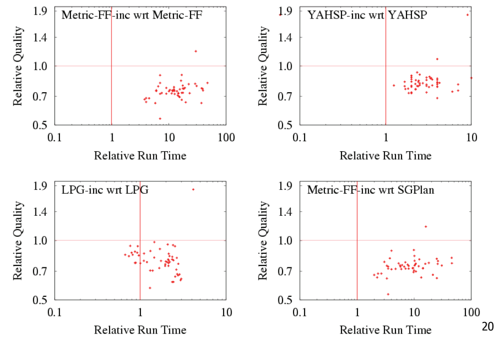
Normalized Performance (cont'd)

Satellite Domain



Normalized Performance (cont'd)

Logistics Domain



Observations

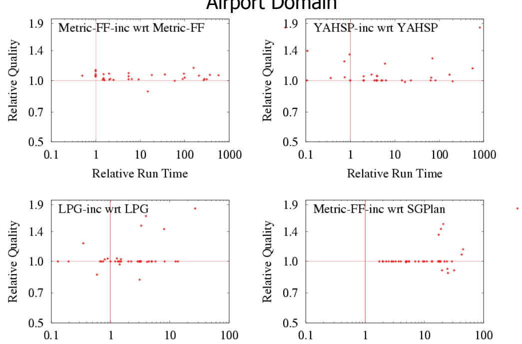
- Generally solve more instances and use less time than the original planners: Metric-FF, LPG, YAHSP
- Incremental planning benefits the most from domains with little goal interactions
  - Airport domain: impossible to invalidate previously achieved "Take-off" subgoals
- Produce longer plans on domains when using improper subgoal order
  - Extra actions due to lots of goal invalidations
  - Likely to occur in domains with intensive goal interactions, e.g. Satellite and Logistics
  - Incremental planning with optimal order can improve run time without sacrificing quality

Redundant-Execution Scheme

- Evaluate two alternative subgoal orders
  - First relaxed-plan order and then original order
    - Restrict the total time spent to 30 minutes
  - Significant improvement in quality over the non-incremental version
    - One of the orders may lead to shorter plans
    - Better solution quality even in domains with intensive goal interactions
  - More run time than evaluating one subgoal order but still less than non-incremental version

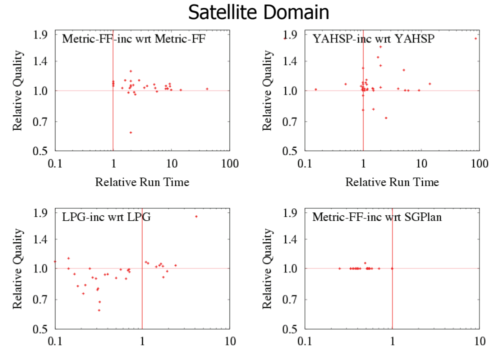
Normalized Performance

Airport Domain



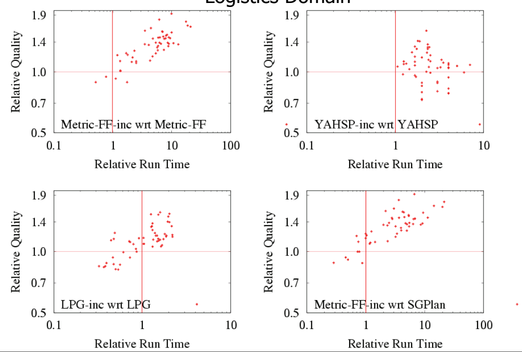
Normalized Performance (cont'd)

Satellite Domain



Normalized Performance (cont'd)

Logistics Domain



Conclusions

- Incremental planning framework for any basic planner
  - Relaxed-plan ordering to reduce number of invalidations
  - Dynamic reordering to avoid dead ends in search
  - Dynamic grain size to achieve good performance
- More instances solved and significant reductions on run time when compared with original planner
- Improvements on both solution quality and run time when using a redundant-execution scheme

Subgoal Ordering 26