e-Merge-ANT

Stephen Fitzpatrick
Cordell Green
Lambert Meertens

Kestrel Institute, Palo Alto, California
http://www.kestrel.edu/HTML/projects/ants

ANTs PI Meeting, Seattle, WA
9-11 May 2000

Outline

- Conceptual architecture for dynamic sensor networks
- Scheduling in the ANTs challenge problem
  - Challenge problem models
  - An iterative repair scheduler based on conflict detection
  - Schedule precompilation through regimes
- Issues
  - Addressing real-time constraints
  - Addressing stability
- Conclusion
  - Summary and plans

A Conceptual Architecture for Dynamic Sensor Networks

4

Introduction

- Networks of sensors and processors
  – monitoring ‘real world’
- Develop an abstract architecture
  – for distributed, real-time resource allocation
  – model specific components to support analysis
- Support ANTs challenge problem
  – common language for participants
  – standard definitions
  – framework in which projects can position themselves
  – not tied to challenge problem

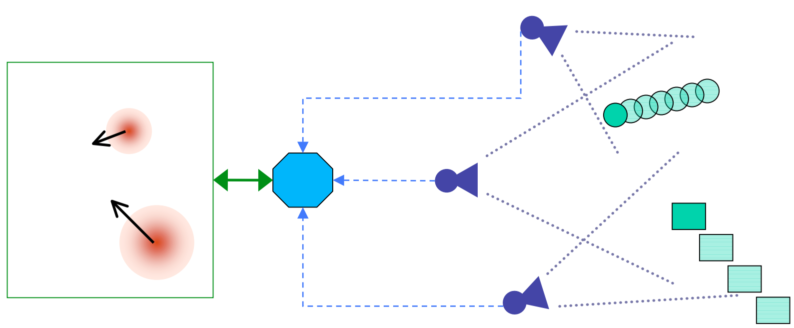
Status

- Initial architecture for sensor networks
- Glossary of scheduling terms
- Formal definitions of sorts and operators for scheduling
- Informal classification of scheduling algorithms
- Detailed definitions for challenge problem

- For your reading pleasure:
  - an overview of the above items is contained in appendices of this presentation
  - detailed documents are now available, and more will become available, on Kestrel's web site
    http://www.kestrel.edu/HTML/projects/ants

Requirements

- Objectives of a dynamic sensor network:
  - Maintain a model of an evolving, real-world environment
  - by measuring certain aspects of the environment
  - and evaluating the sensor data
  - to generate corrections to the model



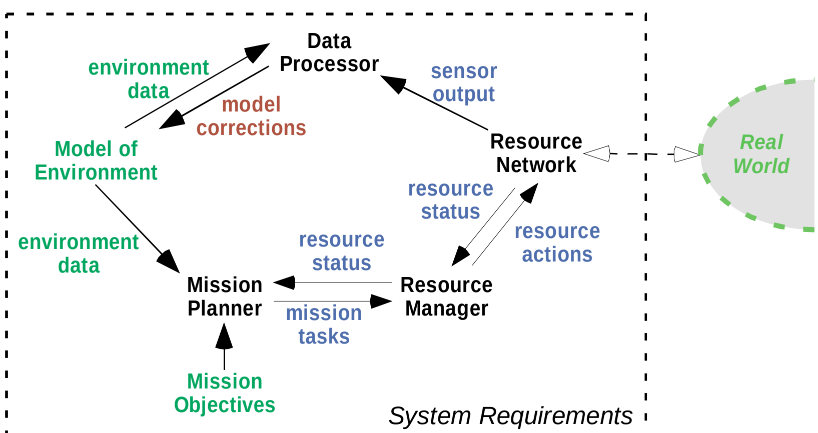
Requirements (cont.)

7

- Requirements define overall functionality of the system
  - e.g., probability of detection and false alarm
  - e.g., required accuracy of tracks
  - e.g., mean time between failures
  - e.g., limit on power/energy of EM emissions
- Outside scope of consideration
  - considerations for EW analyst/engineer

Initial Architecture

8



Model of the Environment

- Defines observable properties of the environment
  - sensor based
  - e.g., EM signal in 1 degree cones
- Defines features of the environment that can be inferred from observations
  - e.g., targets and their states (position, velocity)
- Associates confidence levels with features and states
  - e.g., 95% confident that target lies within certain volume
- Maintains a timed history of observations and features
  - e.g., target tracks
- Interpolates/extrapolates features' states

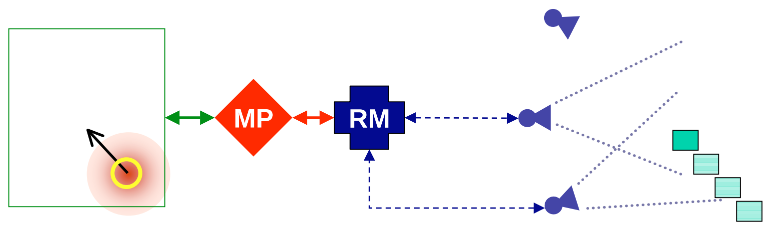
Mission Objectives

- Define desired properties of features and states
  - e.g., confidence level must remain above some level
- Define constraints on how system operates
  - e.g., rate at which power consumption is penalized
  - e.g., rate at which radio communication is penalized

11

Mission Planner

- Translates mission objectives into mission tasks
  - based on the current model of the environment
  - e.g., take a reading on sensor A at angles (θ,φ) at time T
- In the challenge problem, mission tasks are requests from tracking algorithm for further sensor readings
  - includes task priorities, dependencies
- Coupling of mission planner with resource manager?
  - planner may use information about status of resources



12

Resource Manager

- Translates mission tasks into resource actions
  - resources are sensors, processors, communication, etc.
  - schedules the actions
    - including auxiliary actions, such as communication
  - transmits instructions to resources
- Example:
  - sensor A is to emit a beam over some time period
  - and communicate results to controller using channel 4
- Resource manager “optimizes performance”
  - achieves **some** mission tasks as well as possible
  - reduces resource consumption
  - mission objectives define terms for balancing achievement against consumption/cost

Resource Network

- The physical sensors and software interface
- Executes actions generated by resource manager
- Feeds sensor data to data processor
- Feeds status data to resource manager

Data Processor

- Evaluates sensor data
  - measurements are combined with the current model of environment
- Computes corrections to current model
  - e.g., introduces new targets
  - e.g., updates track of existing target
- Data fusion
  - data from multiple sensors are combined to produce ‘best estimate’ of the real world

Scheduling in the ANTs Challenge Problem

Introduction

- Input: measurement requests
- Output: schedule of resource actions
- Schedule the actions of the resources to:
  - optimize achievement of requested measurements
  - reduce usage of consumable resources (energy)
  - reduce EM output
- Time scales:
  - measurements – order of 1 second
  - target observeability – order of 30 seconds
  - target predictability – high in initial scenario
    - In real life?
  - communication?

Status

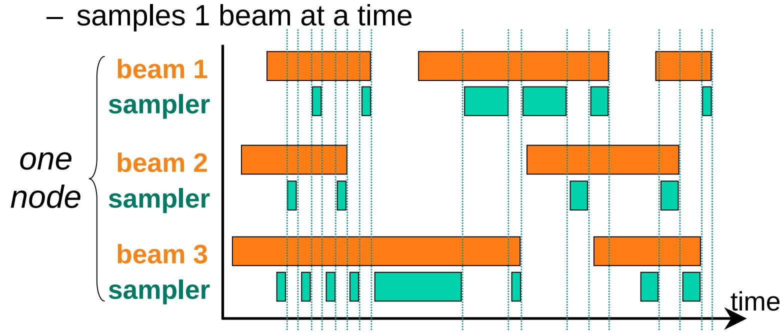
- Have developed formal models for challenge problem
  - for tasks, resources, reservations, schedules, constraints, metrics
- Have investigated the behavior of a preliminary rescheduling algorithm
- Have investigated precompilation of schedules
  - for rapid, real-time response

Challenge Problem Tasks and Resources

- Tracking measurement request
  - node, beam, time, duration, mode, priority
- Background measurement request
  - node, beam, time, duration, mode, priority
- Resources per node
  - 3 beams, 1 sampler, 1 transmitter, 1 receiver, 1 power supply
- Global resources
  - 8 shared communication channels
- Observation: measurement requests uniquely determine radar beam
  - Should there be flexibility to switch request to different node?

Resources: Beams and Samplers

- Three beams per node
  - Each beam independently controlled
  - 1 second warm-up time
  - major power drain
- Single sampler per node
  - samples 1 beam at a time



Resources: Beam and Sampler Constraints

- Beam and sampler reservations are exclusive

op precedes?: Reservation, Reservation → Boolean
def precedes?(p,q) = completion-time(p) < start-time(q)
op disjoint?: Reservation, Reservation → Boolean
def disjoint?(p,q) = precedes?(p,q) ∨ precedes?(q,p)

exclusive-beam-reservations: Hard-Constraint
    = consecutive(beam-resources, disjoint?)
exclusive-sampler-reservations: Hard-Constraint
    = consecutive(sampler-resources, disjoint?)

- Sufficient beam warm-up time

def sufficient-beam-warm-up?(s: schedule) =
 ∀(b∈beam-resources(s), p∈reservations(s,b))
  ¬ ∃(m∈sampler-resources(s), q∈reservations(s,m))
    b=beam(task(q)) ∧ overlap?(p,q) ∧
    start-time(q)<start-time(p)+warm-up(b)
beam-warm-up: Hard-Constraint = global(sufficient-beam-warm-up?)

21

Resources: Communication

- Communication channels
  - exclusive reservations
  - finite latency and bandwidth
- Communication task defines
  - channel, sender, receiver, message
- Constraint: sufficient communication time reserved

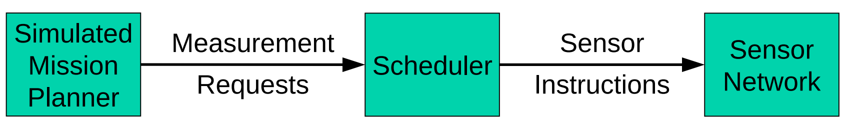
for channel c: processing-time(c,communication-task(c,s,r,m))
    = latency(c,s,r) + data-size(m)/bandwidth(c,s,r)

op sufficient-processing-time?: Reservation -> Boolean
def sufficient-processing-time?(r)
    = duration(r) > processing-time(resource, task(r))

sufficient-communication-times: Hard-Constraint
    = pointwise(channel-resources, sufficient-processing-time?)

22

An Experiment with An Iterative Repair Scheduler



- Investigate scheduler to get a feel for domain
  - some missing information – educated guesses
- Simplifications:
  - ignore communication (not a bottleneck?)
  - ignore power (no impact on feasibility)
- Two types of task (distinct):
  - tracking measurement
  - background measurement
  - node, beam, start time, duration, priority

Experiment: Scheduling Objectives

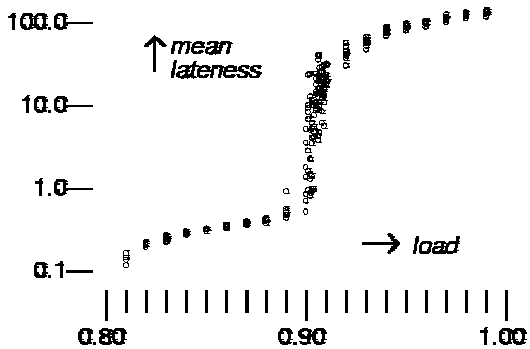
- Assumed constraints
  - radar nodes can perform only one task at a time
  - durations are hard constraints
  - start times are soft constraints
    - but the *order* of task execution is determined by the requested start times
- Objective: minimize weighted mean lateness

$$\sqrt{\frac{\sum_r w(r).[\text{start-time}(r) - \text{due-date}(r)]^2}{\sum_r w(r)}} \qquad r \in \text{reservations}$$

Experiment: The Algorithm

- Iterative repair
  - locate hard constraint violation
    - caused by overlapping reservations on single node
  - linearize the reservations
  - translate the reservations' start times to minimize objective function
    - i.e., produce lowest total deviation from due dates
  - repeat
- Run-time negligible

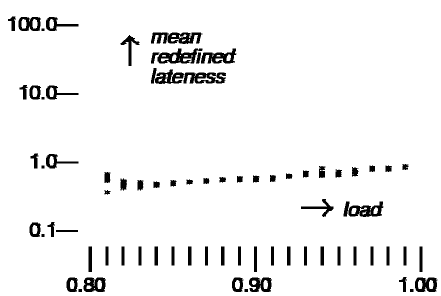
Experiment: Parameters and Results

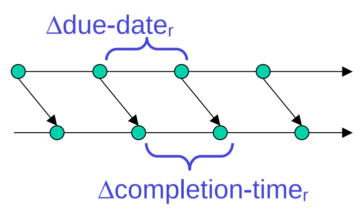


load = $\sum_k \text{duration}_k \cdot \text{frequency}_k$

- Parameters
  - Tracking measurements
    - duration: 2.0 seconds
    - every 2.5 seconds (approx.)
  - Background measurements
    - duration: 0.5 seconds
    - frequency varied to set load

Experiment: Redefined Lateness



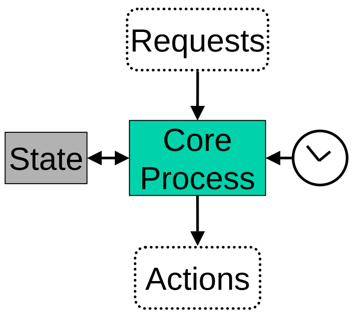


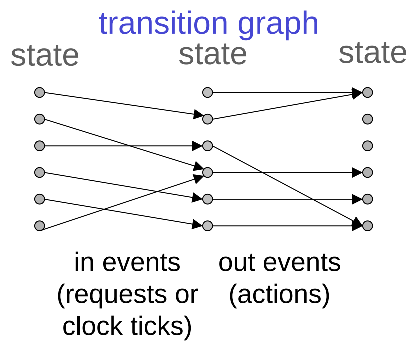
- New assumption: periodic measurements
  - regularity of measurements important
  - redefine lateness
  - $\text{lateness}_r = \Delta\text{completion-time}_r - \Delta\text{due-date}_r$
- Need domain knowledge to select appropriate metric

27

Schedule Precompilation

- Objective: fast schedule look-up
  - in scenarios having demanding response times
  - identify *scheduling regime*
  - look up appropriate scheduling strategy
- Scheduling strategy is determined by transition graph





28

Precompilation Process

- Table of optimal strategies precompiled off-line
  - definition of optimality includes anticipated distribution of future events
- Assume a finite state space
  - can be obtained by discretization
- Theoretically it is possible to enumerate all and select best
- Practically: useful spaces very large
  - Use branch & bound or constraint propagation techniques to reduce size?
  - Precompile solutions for coarse space
    - Refine at run-time with rapid fast improvement method

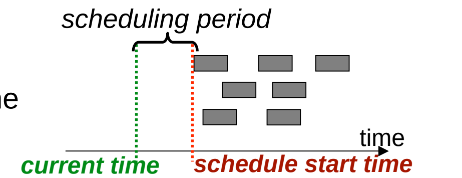
Issues

Addressing Real Time Constraints

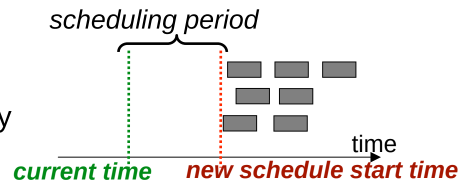
- Two approaches proposed
  - two tier scheduling
  - precompilation
- Two tier scheduling
  - First tier uses an *anytime scheduler* to schedule sensor actions
    - sequence of schedules computed over time
    - each schedule better than preceding schedules
  - Second tier allocates time to first tier scheduler
    - needs to stop first tier scheduler when schedule execution must begin

Available Scheduling Time

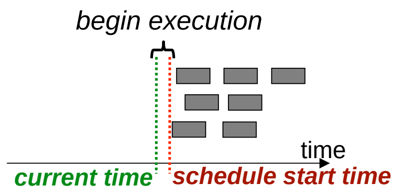
- Scheduling deadline
  - a schedule's start time is the earliest of its reservations' start times



- Deadline is dynamic
  - as anytime scheduling proceeds, the start time may shift



- Exploit time before deadline
  - monitor current time against schedule start time
  - begin execution when the two times converge



Schedule Time versus Execution Time

- Quality of schedule should improve as first tier scheduler runs
  - allocate as much time as possible to scheduling
  - postpone the start time
- But, in a dynamic environment, a schedule should leave room for change
  - execute tasks as early as possible
- Define tradeoff between schedule improvement and robustness
  - Can measurement process be viewed as anytime?
  - If so, can we use existing techniques/tools to compose anytime processes?

Real-time Response Using Precompilation

- In a scenario where precompilation is feasible
  - response time is very fast – no real-time problems
- In a scenario that is too complex for precompilation
  - derive a crude discretization for which precompilation is feasible
  - use look-up to obtain an initial, crude schedule
  - complete crude schedule using constraint propagation
  - refine completed schedule using local search
  - objective is to ensure that crude schedule places search in a good neighborhood so that it rapidly converges to an optimum

Addressing Stability

- Continuity-guided rescheduling
  - can adapt local and global scheduling algorithms to rescheduling
  - local search
    - use old schedule as starting point for repair
  - global search
    - can incorporate an explicit cost of change into quality metrics
    - can influence search guidance metrics so that the decisions that lead to the old schedule are given extra weight when constructing new schedule

35

Addressing Stability: Schedule Spaces

- Schedule spaces
  - some global search algorithms can produce spaces of feasible schedules
  - allows continuous change under small perturbations in input/environment
- Sub-optimality/schedule slack
  - optimal schedules tend to be fragile
  - Deliberately aim for sub-optimal resource usage to ensure some slack in schedule?
  - Can we make this notion precise?

36

Addressing Stability: Explicit Costs

- Explicit costs in large, distributed systems
  - all resource actions, including those undertaken for scheduling, are explicitly costed
  - costs should act as dampening factor in distributed systems
- Convergence
  - anytime schedulers
    - time bounded (ensures limited fruitless activity)
    - monotonically improving schedule

Conclusion

Summary and Plans

Summary

- Have made a start on abstract architecture for distributed resource allocation
- Have made a start on modeling the challenge problem
  - have identified some topics for discussion

Plans

- Model further aspects of distributed resource allocation
- Synthesize schedulers for the challenge problem
  - investigating appropriateness of anytime algorithms
- Demonstration of scalability
  - measure schedule quality as number of tasks and resources increases

Appendix
Sorts and Operators for Scheduling

Overview of Schedule Sorts and Operators

- Tasks: requests for when, where and what
- Resources: constraints on activities
  – task processing times, set up times
- Reservations: task, resource and time period
  – typical constraints: task & resource compatibility, release dates observed, non-overlapping
- Schedule: set of reservations
  – typical metrics: makespan, total weighted tardiness
- Constraints: hard, soft & precedence

Tasks and Resources

- Sort Task
  - ops: type, release-date, due-date, weight
- Sort Resource
  - ops: type, compatible-task?, processing-time, setup-time

Reservations and Schedules

- Sort Reservation
  - ops task, resource, start-time, completion-time, duration, precedes?, overlap?, lateness
  - release-date-observed?, compatible-resource-and-task?, sufficient-setup-time?
- Sort Schedule
  - ops: reservations, consecutive?, makespan, maximum-tardiness, total-weighted-tardiness, complete?

Constraints

- Three classes of constraints:
  - hard, soft & precedence constraints
- Sort Hard-Constraint
  - a hard constraint cannot be violated
  - in general, a hard constraint is an arbitrary boolean function on schedules
    - e.g., all-tasks-scheduled?
  - common classes of hard constraints:
    - pointwise: lifts a single-reservation constraint
      - e.g., release-date-observed?, compatible-resource-and-task?
    - consecutive: lifts a constraint on neighboring reservations
      - e.g., sufficient-setup-time?

Soft Constraints

- Sort Soft-Constraint
  - a soft constraint can be violated, but violation incurs a penalty
  - in general, a soft-constraint can be an arbitrary function on schedules
  - common classes of soft constraints:
    - pointwise: lifts single-reservation test and penalty functions
      - e.g., due-dates-observed? & lateness
    - consecutive: lifts constraint and penalty functions on neighboring reservations
      - e.g., zero-wait? & idle-time

Precedence Constraints

- Sort Precedence-Constraint
  - a precedence constraint is a strict partial order on tasks:
    - anti-reflexive
    - anti-symmetric
    - transitive
  - for tasks p,q
    - if (p,q) is in the precedence constraint
    - p must be completed before q begins

Feasibility

- Given:
  - R, a set of resources
  - T, a set of tasks
  - H, a set of hard constraints
  - P, a set of precedence constraints
- A schedule is feasible if and only if it observes all constraints in H and P

op find-feasible-schedule:
set(Resource), set(Task), set(Hard-Constraint), set(Soft-Constraint)
$\rightarrow$ **Schedule**

axiom feasible
$\forall$(R,T,H,P) $\forall$(h$\in$H) observes?(h,find-feasible-schedule(R,T,H,P))
$\wedge$ $\forall$(p$\in$P) observes?(p,find-feasible-schedule(R,T,H,P))

47

Optimality

- Given also
  - S, a set of soft constraints
  - Q, a metric on schedules
- A schedule is optimal if and only if it minimizes the (weighted) sum of Q and penalties arising from S

op find-optimal-schedule:
set(Resource), set(Task), set(Hard-Constraint),
set(Precedence-Constraint), set(Soft-Constraint), Quality-Metric
$\rightarrow$ **Schedule**

Axiom optimality
∀(R,T,H,P,S,Q) ∀(s':schedule) feasible?(s',R,T,H,P)
⇒total-penalty(s',R,T,P,H,S,Q)
≥ total-penalty(find-optimal-schedule(R,T,H,P,S,Q),R,T,S,Q)

Appendix
Data Fusion Classifications

Reference
Sensor and Data Fusion Concepts and Applications,
Lawrence A. Klein, SPIE Press Vol. TT35,
ISBN 0-8194-3231-8

Data Fusion Levels

As Defined by Office of Naval Technology

- Level 0: source preprocessing
  - e.g., compression, normalization of sensor data
- Level 1: object refinement
  - target track estimation and target discrimination
- Level 2: situation assessment
  - relationships between targets
  - identification of activities
- Level 3: threat assessment
  - capability and intent estimation
  - offensive and defensive analysis
- Level 4: fusion process refinement
  - feedback to sensors and processing
- Challenge problem: levels 0, 1 & 4?

Processes in Track Estimation

- Alignment
  - placing sensor data into a common coordinate system
- Association
  - computing a metric to determine how well measurements and tracks match
- Correlation
  - using association metrics to determine if measurements and tracks correspond to a common object
- Estimation
  - updating target states using the results of correlation
- Cueing
  - feedback to sensor control and processing
- Challenge problem: who is responsible for what?

51

Data Fusion Architectures

- Sensor-level fusion
  - each sensor processes data locally and sends results to a central fusion unit
  - distributes computational load
  - tailors processing to each sensor
  - tightly coupled data processing and sensor control
- Central-level fusion
  - each sensor feeds minimally processed data to a central fusion unit
  - more accurate results
- Hybrid fusion
  - each sensor performs some processing of its data, but also feeds minimally processed data to central unit
- Challenge problem: hybrid fusion?

52

Data Fusion Modes

- Pixel-level fusion
  - data from multiple sensors are fused at the pixel level
  - used in central-level fusion
- Feature-level fusion
  - each sensor's data is processed to produce features which are then combined
  - used in sensor-level or central-level fusion
- Decision-level fusion
  - each sensor's data is processed to produce target tracks and classification data
  - the tracks and classification data are fused
  - used in sensor-level fusion
- Challenge problem: feature and decision level?

Appendix
Scheduling Algorithms

54

Classification of Scheduling Algorithms

- Heuristic algorithms
  - typically based on immediate priority rule (dispatch)
  - quickly compute reasonably good schedules
- Local search algorithms
  - iterative improvement of a complete schedule
  - may get trapped in a local optimum
    - methods exist to attempt to reach global optimum
- Global search algorithms
  - construct globally optimal schedules
  - large search spaces (lots of backtracking)
    - methods exist to trade quality for speed
- Anytime algorithms
  - schedule always available
  - schedule improves as algorithm runs

55

Local Search Variants

- Simulated annealing algorithms
  – occasionally chose to temporarily degrade schedule in hope of escaping local optima
  – frequency of degrading reduces with time
- Tabu search algorithms
  – maintain a list of recent schedule transformations
  – a transformation is forbidden if its reverse is on the list
  – when a transformation is made, its reverse is placed on the list and the oldest entry is removed
  – attempts to avoid cycles
- Genetic algorithms
  – maintain a population of schedules
  – delete poor schedules from population
  – produce new schedules by cross-breeding and mutation

56

Global Search

- Operate on spaces of schedules
  – spaces iteratively refined by making choices
  – e.g., assigning a particular task to a particular resource
  – backtracking is typically used
    - because a choice may ultimately prevent the construction of a feasible, complete schedule
  – relaxation may be used instead of backtracking
    - when all remaining choices result in infeasible schedules
    - relax some of the constraints on the remaining tasks
    - penalty may be incurred for relaxation
- Typically produce optimal schedules
  – may sacrifice optimality for speed (e.g., relaxation)

Global Search Variants

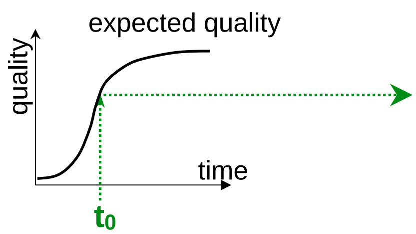
- Pruning
  - use weakened constraints to quickly detect choices that will produce no feasible, complete schedule
- Branch & bound
  - use lower bound computations on schedule quality
  - quickly determine if a choice cannot produce an optimal schedule
- Priority search
  - at each iteration, rank choices according to some heuristic
  - investigate only the most promising choices

Anytime Algorithms

- Interruptible
  - can be stopped at any time and will return a schedule
- Monotonic improvement
  - the longer the algorithm is allowed to run, the better the schedule
- Performance profiles
  - statistical characterization of quality of schedule against run time

Performance Profiles

- Expected quality specified as a *performance profile*
  - maps time onto a probability distribution of quality
  - may also depend on characteristics of input data



performance profile

probability	quality
15%	mean + 2
20%	mean + 1
30%	mean (expected value)
20%	mean - 1
15%	mean - 2

probability distribution of quality at time t_0

Potential Anytime Schedulers

- Local search algorithms are commonly converted into anytime algorithms
  - extract single search step from local search
  - use as core for anytime algorithm
- Potentially, global search algorithms may be used
  - in anytime domain, every schedule can be assigned a quality
  - in backtracking, even a partial schedule that cannot be extended into a complete schedule has a value
    - it is a feasible schedule for those tasks that have been scheduled
- Retain best schedule found as search tree is explored
  - regardless of (eventual) completeness

61

Branch & Bound Anytime Scheduler

- Branch & bound with tolerance
  - parameter can be adjusted to set tolerance for sub-optimality (e.g., within 10% of optimal)
    - ignores branches whose lower bound is not sufficiently better than the current best solution
  - lower tolerance gives better result but (typically) increases the size of the space searched
- Begin with tolerance high
  - seed algorithm with scheduler produced heuristically
- Reduce tolerance on successive iterations
  - use schedule computed as seed for next iteration
- Can iterations be performed sufficiently quickly?

62

Priority Search Anytime Scheduler*

- Priority search algorithm:
  - construct a search tree
  - for each node, rank all branches (without searching)
  - search only *W* most promising branches
    - *W* is the search width
  - does not guarantee optimality
- A low width reduces search time
- A high width is likely to find better result
- Begin with a narrow search width
  - e.g., *W*=1
- Widen search on successive iterations
  - retain best schedule found

**Priority search is commonly known as beam search. I have renamed it to avoid potential confusion with "beam" in the sense of a radar beam.*