Scheduling Background: Glossary

Stephen Fitzpatrick (fitzpatrick@kestrel.edu)
Kestrel Institute, 3260 Hillview Avenue, Palo Alto, CA 94304
http://www.kestrel.edu/home/projects/ants/

This document is one of a set that attempt to bring together background information on scheduling. In writing these documents, we have not attempted to be comprehensive, but rather have concentrated on information useful for Kestrel's ANTs project. The contents of the set of documents are:

1. Glossary
   Informal definitions of terms generic to scheduling.
2. Specifications
   Formal definitions of sort and operations for scheduling.
3. Algorithms
   Informal descriptions of classes of algorithms used in scheduling.

If you do not have any of the other documents in the set, you should be able to find them at our web site: http://www.kestrel.edu/home/projects/ants/scheduling/

Caveat: this is a work in progress.

1 The Glossary

Closure time (of a schedule)
Same as "makespan" – the latest completion time of all the reservations in a schedule. If the schedule starts at time 0, then the closure time is also the schedule's duration.

Compatibility constraints (between resources and tasks)
A relation between tasks and resources defining which tasks can be accomplished by which resources.

Example: an acoustic sensor may not be able to perform target tracking.

Completion time (of a reservation or corresponding task)
The time at which a reservation/task is scheduled to be completed.

Constraint
A rule that limits the schedules that are considered acceptable.

Example: a sampler can perform only one measurement at a time.

Constraint, hard
A constraint that cannot be violated, typically reflecting a physical limitation of a resource.

Example: the duration of a communication task must be long enough to allow for the communication system's latency and finite bandwidth; any schedule that violates this constraint cannot be performed.

Constraint, soft

A constraint that can be violated, typically incurring some penalty.

Example: a task may be scheduled to be completed after its due date; a penalty may be incurred according to the tardiness.

Due date (of a task)

The latest allowed completion time of a task.

Example: a measurement must be completed by time T.

Feasibility (of a schedule)

A feasible schedule is one in which no precedence constraints or hard constraints are violated.

Gantt chart

A graphical display of a schedule.

Example: a horizontal bar chart showing resources on the vertical axis against reservations, with time on the horizontal axis.

Makespan (of a schedule)

Same as "closure time" - the latest completion time of all the reservations in a schedule. If the schedule starts at time 0, then the closure time is also the schedule's duration.

Maximum tardiness (of a schedule)

The largest tardiness of all the reservations in a schedule.

Precedence constraints (between tasks)

A strict partial order constraining the order in which tasks are begun/completed.

Example: a measurement task may have three sub-tasks – send instructions to a remote node, perform the measurement, communicate the data back to a central processor – that must be executed in order.

Processing times (properties of resource and task types)

How long it takes to accomplish a given task (or type of task) on a given resource.

Example: to send N bytes over a communication channel takes time (L+N/B), where L is the latency and B is the bandwidth.

Quality (of a schedule)

Some metric in a partial order defined on schedules.

Example: the total weighted tardiness.

Release date (of a task)

The earliest time at which any resource can start to process a given task.

Example: if a measurement is scheduled to be completed at time T, then the task of communicating the results has a release date of T.

Reservation
A triplet indicating that some task is scheduled to be executed on some resource over some time period.

Example: sensor A is to perform measurement task M beginning at time T_s and ending at time T_e.

Resource
An entity capable of accomplishing tasks.

Example: a radar beam.

Setup times (properties of resource and task types)
For a given resource, the time to prepare the resource between tasks.

Example: a radar node is to perform an active scan task after a passive listening task and requires a time T to warm up the beam.

Schedule
A set of reservations defining the processing of a set of tasks on a set of resources.

Tardiness (of a reservation)
The amount of time (non-negative) by which a task's completion time follows its due date.

Task
An element of work to be accomplished.

Example: take a measurement on sensor A, direction (θ,ϕ), at time T, for duration D.

Total (weighted) tardiness (of a schedule)
The (weighted) sum of the tardinesses of all the reservations in a schedule.

Weight (of a task)
The relative importance of a task, in some total order, typically the natural numbers or reals.

Example: a tracking task may have weight 9 whereas a background sweep task may have weight 5, indicating that the tracking task is more important.

1.1 REFERENCES

Scheduling: Theory, Algorithms and Systems, Michael Pinedo,
Prentice Hall 1995, ISBN 0-13-706757-7

Scheduling Background: Algorithms

This document is one of a set that attempt to bring together background information on scheduling. In writing these documents, we have not attempted to be comprehensive, but rather have concentrated on information useful for Kestrel's ANTs project. The contents of the set of documents are:

4. Glossary
   Informal definitions of terms generic to scheduling.
5. Specifications
   Formal definitions of sort and operations for scheduling.
6. Algorithms
   Informal descriptions of classes of algorithms used in scheduling.

If you do not have any of the other documents in the set, you should be able to find them at our web site: http://www.kestrel.edu/home/projects/ants/scheduling/

Caveat: this is a work in progress.

2 Scheduling Algorithms

A scheduling algorithm attempts to assign tasks to resources at time periods, such that the constraints on the tasks and resources are observed, or at least minimally violated. Typically, some attempt is made to optimize the schedule according to some quality metric. In general, scheduling is NP-hard, so practical methods for achieving good run times in particular problem domains are of interest.

2.1 Heuristic Algorithms

Heuristic scheduling algorithms typically attempt to achieve reasonably good, feasible schedules by assigning tasks to resources according to an order based upon some *criticality* measure. For example, tasks may be ordered according to the ratio of their availability duration (due date less release date) to their processing time; or resources may be ordered according to their total load (which changes as the algorithm schedules tasks). Typically, heuristic algorithms do not revise task assignments even if the schedule turns out to be poor.

2.2 Local Search Algorithms

A local search algorithm computes on a single candidate schedule that is already complete – in the sense that it contains reservations for all of the tasks that are to be scheduled – but that may be infeasible or sub-optimal. A local search algorithm typically performs a transformation on the candidate schedule to produce a better neighboring schedule, and iterates this process until a satisfactory schedule is produced or a local

optimum is reached. An example of a transformation is shifting a reservation forward in time to correct a due-date violation. It is not guaranteed that the final schedule is globally optimal. Local search algorithms are also called iterative repair algorithms.

The primary considerations in designing a local search algorithm are:

- generation of an initial seed schedule;
- generation of the neighbors of a candidate solution;
- selection of the best neighbor;
- avoiding being trapped in unsatisfactory local optima.

2.2.1 Simulated Annealing

In standard local search algorithms, the algorithm only progresses from some candidate schedule to a *better* neighboring schedule. As a result, the algorithm may become stuck in a local optima and fail to find a global optima. Simulated annealing is a variant of local search that tries to escape from local minima by occasionally progressing to neighbors that are *worse* than the current candidate schedule. At each iteration of the search, each worse neighbor of the current candidate schedule has a (low) probability of being accepted as the next candidate schedule. The probabilities are assigned according to some function of: (1) the qualities of each neighbor relative to the quality of the current candidate; (2) the length of time the algorithm has been running. The longer the algorithm runs, the less likely the worse neighbors are to be accepted. Eventually, probability of progress to a worse neighbor becomes negligible and the search settles into a local optimum.

2.2.2 Tabu Search

In tabu search, the search algorithm may progress from a current candidate solution to any neighboring solution – better or worse – provided only that the transformation to the neighbor is not contained in the current *tabu-list*. The tabu-list is a fixed length list of schedule transformations that are currently forbidden; when a transformation is applied to the current candidate schedule, that transformation is added to the head of the list and the tail of the list is removed. The intention is to try to prevent cycling in the search process.

2.2.3 Genetic Algorithms

Genetic algorithms operate on finite-sized populations of candidate schedules. At each iteration of the algorithm, relatively poor schedules are removed from the population and are replaced with new candidate schedules generated by: (1) applying mutations to individual schedules in the population; (2) applying cross-over operations to pairs of schedules in the population.

2.3 Global Search Algorithms

Global search algorithms find feasible or globally optimal schedules by searching through schedule spaces. Typically, a search tree is constructed in which each node represents, say, a task that is to be scheduled and each branch from that node represents a choice of to which resource the task is assigned. Each partial schedule constructed in this way is required to be feasible. If a node is reached for which no choice of task and resource produces a feasible schedule, then back-tacking occurs.

For finding a feasible schedule, the algorithm terminates when it has constructed a feasible schedule and no tasks remain to be scheduled. For finding an optimal schedule, each such complete, feasible schedule is compared using some quality metric and the best one is returned; this is a globally optimal schedule.

The primary considerations in designing a practical global search algorithm are:

- reducing the size of the search space;
- selecting a good order for considering the siblings branches at each level (since this may affect the ability to reduce the search space).

2.3.1 Pruning

Global search algorithms may use pruning to reduce the size of the search space. For example, the constraints that a feasible schedule must satisfy may be weakened to produce necessary constraints on partially constructed schedules. If the weakened constraints can be quickly computed at each node in the search space, they can be used to prune off branches that cannot produce feasible schedules.

2.3.2 Branch & Bound Algorithms

Global search algorithms may also use the schedule quality metric to prune off search branches. For example, assume that a feasible schedule is required that maximizes some metric. It may be possible to compute an upper bound on the metric for any given branch in the search tree, such that no feasible schedule contained in that branch has a metric that exceeds the upper bound computed for that branch. If a branches upper bound does not exceed the metric of some schedule that has already been constructed, then the branch cannot contain an optimal schedule and the branch can be pruned. If a heuristic scheduler is available that produces reasonable schedule, then it can be used to seed the branch & bound algorithm.

If an approximately optimal solution is sufficient, a more liberal pruning policy can reduce the search space still further: a branch is pruned off if its upper bound does not exceed, by some defined tolerance, the highest value of the metric found so far. For example, if the tolerance is set to 10%, then the schedule returned by the algorithm is guaranteed to be within approximately 9% of optimal.[1]

2.3.3 Beam Search

Beam search is a variant of branch & bound in which, at each node in the search tree, the branches are ranked according to some metric and all but the best *w* branches are pruned, where *w* is called the beam width. Thus, beam search reduces the search space but sacrifices optimality. It is important that the ranking function selects branches that ultimately produce high-quality schedules, but it is also important that the ranking process not be too computationally expensive. Thus, the ranking process may be preceded by a filter that quickly discards most of the branches, leaving a relatively small number of branches to be ranked through a more thorough processes.

[1] If the algorithm at tolerance T% finds a schedule with metric M, then the optimal schedule has a metric no larger than M×(1+T/100).

2.3.4 Relaxation

Relaxation is a technique that can be used to avoid backtracking, at the expense of producing a schedule that is approximately feasible. At each node in the search space, a single branch is selected that does not *immediately* lead to an infeasible schedule. The search algorithm follows that branch and never leaves it. Eventually, either: (1) a feasible schedule is created in which all of the tasks have been scheduled, and this schedule is returned as the result of the algorithm; (2) a node is reached in which some tasks remain to be scheduled and from which every branch immediately produces an infeasible schedule. In the latter case, rather than backtracking, the constraints on the unscheduled tasks are relaxed and scheduling continues. For example, a task may be relaxed by extending its due date. Typically, there is a pernalty associated with relaxation.

Thus a search algorithm that uses relaxation may produce sub-optimal schedules (where the hard concept of infeasibility is replaced with the soft concept of penalties) but it typically runs in time that is approximately linear in the number of tasks.

2.4 ANYTIME ALGORITHMS

A scheduling algorithm is classified as being anytime if: (1) it can be allowed to run for an arbitrary amount of time, unknown in advance, and return a valid schedule when interupted; (2) statistically, the longer the algorithm is allowed to run, the higher the quality of the schedule it produces. An anytime algorithm may be characterized by a performance profile that maps running times into probability distributions of schedule quality. The distributions may also depend on some characteristic of the input data, in which case the profile is called a conditional performance profile.

Typically, an anytime scheduler works by producing some initial schedule, maybe using a heuristic algorithm, and then iteratively refining the schedule. The iterative process may actually temporarily degrade schedule quality, in which case the algorithm may also retain a copy of the best schedule found so far, to be returned as its answer if it is interupted.

Local search algorithms can generally be converted into anytime algorithms simply by extracting the iterative search process and using it in a local search schema that provides for interupt capabilities, etc. It may also be possible, although perhaps not useful, to convert a global search algorithm into an anytime algorithm: the anytime version simply records the best partial schedule constructed so far and returns that as its answer if interupted. A partial schedule produced by a global search algorithm will not satisfy all of the constraints (e.g., it will not contain assignments for all of the tasks that were to be scheduled) but it presumeably will nevertheless be of some value.

Furthermore, global algorithms such as branch & bound with tolerance and beam search are parameterized and it may be possible to construct a useful anytime algorithm by iterating such an algorithm wth successive values of the parameters such that: (1) for the initial parameter values, the algorithm runs quickly; (2) for successive parameter values, the algorithm typically runs more slowly; (3) the schedule found by the algorithm is likely to increase in quality for successive values of the parameter. During this process, the best schedule found so far is stored to be returned as the answer if the algorithm is interupted.

For example, a branch & bound algorithm can be run with an initially high tolerance, say 50%, to quickly produce a sub-optimal schedule. Successive iterations of the algorithm may reduce the tolerance, perhaps finally to zero. Because the size of the search space increases as the tolerance is reduced, successive iterations will (likely) take longer to run. For a beam search algorithm, the beam width may be initially set low and increased on successive iterations.

2.4.1 Contract Algorithms

A contract algorithm is one that is informed on initiation of the time for which it will be allowed to run. The algorithm must produce a valid result when it is terminated/it terminates itself at that time. Like an anytime algorithm, a contract algorithm will produce better schedules the longer it is given to run; unlike an anytime algorithm, it must be told in advance how long it will have to run, and it may not return a valid result if it is interupted before that time.

2.5 RESCHEDULING

TBD