Techniques Common to Most Methods of Schedule Optimization

By Steve Morrison, Ph.D. 1997 Info@MethodicalMiracles.com 214-769-9081

There are a number of issues in schedule sequencing regardless of the solutions techniques applied. Every method has an initialization procedure, some methods benefit from pre-conditioning, and heuristic methods have a stopping criteria. For a few methods, restart criteria and methods are important, too. Finally, many methods, with some modifications, are suitable to run on computers with parallel processors.

Approach

The same approach can have varying degrees of success with different problems and different formulations of the same problem. The objective function to use is not an unalterable constant, but can often be modified to improve the speed of solution. One can change the problem characteristics to scheduler for a fixed number of future batches instead of a fixed time. One can allow batches not to be scheduled, with a prize-collecting penalty, instead of requiring the scheduling of all batches. Alternately, one can allow multiple batches of the same kind of job, with a lagrangean penalty, for all batches. The problem formulation options can be more limited, however, if a rigorous optimum is required. Even for problems without a rigorous optimum requirements, some methods are less useful if an optimum bound is desired. One can seek to maximize throughput versus minimize makespan. There are different ways to separate the continuous and discrete aspects of a problem. Discrete aspects can be separated from each other through relaxations, and all discrete aspects can be solved for a subsection of the problem with problem decomposition.

Initialization

For some methods, such as tabu search and simulated annealing, the cost of the initial solution does not matter much; for other methods, such as branch and bound, the initial cost value is very important. Thus, there can be merit in using a quick heuristic method, followed by branch and bound to get the final solution.

Another important part of initialization is pre-conditioning the costs. Heuristic methods typically need to choose among the best appearing candidates, and anything that can improve the probability of a best-appearing candidate being actually the best candidate will usually help these methods. The branch and bound method can perform well or perform poorly, depending on the tightness of the bounds. This author proposes a pre-conditioning procedure based on both setup costs and on setup times to improve both candidate selection and branch pruning. In assignment problems, looking a row of costs, in isolation, is not helpful in estimating which elements are the most likely candidates for being assigned; one has to look at all the cost matrix. White [1994]

has a method for polynomially solving quadratic assignment problems with a special structure. This can also apply to assignment problems if they have the special structure of costs such that

$$a_{ij} = u_i + v_j \qquad \text{for } i \neq j, \qquad \text{for rows i and j.}$$

Admittedly this is not often the case for either assignment problems or quadratic assignment problems, but putting a zero in each row and column using matroid subtractions to precondition the cost matrix is at least superior to the raw cost matrix. Philosophically, the reason this helps is that from a branch and bound perspective there is no cost for what appears to be the best candidate. An even better solution than that is to solve the assignment problem, and use that as the cost matrix. Philosophically, the reason this improves the bounds is that the assignment problem matrix at the solution not only gives a solution, but also tends to have the lowest costs for the best candidates and the highest costs for the worst candidates. The following paragraph is an analogy which will relate to the reasons for this heuristic.

A card deck with an unknown number of playing cards, and an unknown distribution of cards is shuffled. Five people each have two cards, and the objective of the game is to guess whose cards have the lowest sum. There is no information to favor one over the other, so the probability of the first person having the lower sum is 0.2. Now suppose the first person turns his first card over, and he has a low number, such as 3. If one guessed that the average of the first cards was around 6, then one would be somewhat more likely to guess the first player assuming there is no known correlation between the first card and second for each player. However, if every player turned over the first card, then one should guess the player with the lowest visible card. One is normalizing all the first cards, and using the card with the normalized value of zero.

For an n-job jobshop scheduling problem with constraints, the probability that job j follows job i is i/n, because it could follow n - 1 jobs or it could be the first job. Of course job j will not follow if it does not have the greatest total benefit, and it will follow job i if it does. Now by applying the separation of assignments philosophy, the total benefit is the sum of the local benefit plus the global benefit. One can think of these two costs as two playing cards, with the turned over card being the local benefit. With the initialization procedure, on can think of using the lowest normalized cost to guess the best candidate. For ten job problems, this author has seen time savings of 5 to 10%.

One can use a similar procedure to minimize the rows of the setup times. However, the removed constant time must be added to the duration of the job and the earliness and lateness dates will need adjustment. One can also minimize the columns of setup times. Both of these minimizations become problematic though, when the job durations are permitted to change. This can occur if one is allowed to speed up a job at the expense of yield or quality.

Many times the scheduling problem is such that any simplifications or special problem structure are known in advance. However, for jobshop scheduling problems the region where simplified solution techniques still give globally optimal answers can vary from schedule to schedule. Jobshop scheduling is a simple problem when no jobs will ever be early or late, or all

lateness and earliness penalties are constant. Jobshop scheduling is also much simpler than in the general case if all jobs are late regardless of the sequence. Thus for some scheduling codes, it makes sense to have a quick pre-scheduling test to determine if the schedule will match a problem structure and be more amenable to a particular solution algorithm.

Stopping Criteria

For rigorous methods, the stopping criteria is usually simply after trying every permutation that could possibly be an optimum. For heuristic methods, the stopping criteria parallel the stopping criteria for continuous optimization. Dennis and Schnabel [1993] devote an entire chapter to stopping, scaling and testing. Their continuous scaling is closely analogous to pre-conditioning for integer programming problems. They mention five criteria that are relevant here, and the author adds a sixth.

1. Global optima: If the best solution possible is known from the global lower bound, then stop.

2. Local optima: Either quit because the method is unable to get out of a local optima, quit because it is known there is only one optima, or restart when the algorithm finds a local optima.

3. Stalling: This occurs when there is no more improvement or the improvement is very small for a certain number of moves. Stalls can not only be a stationary point, but in Tabu search and other methods, a stall can be a limit cycle of solutions.

4. No more time: The computer can give the best solution after a preset amount of elapsed time or iterations. This limit can be varied by the user. In some genetic algorithm and simulated annealing codes, the limit can be predetermined by the algorithm.

5. Patience: It is good for the user to be able to stop the program at any time, returning the best solution so far. After stopping, the user might want to restart the program at the point where he last stopped it.

6. No Feasible solution: It is good to rapidly determine when there is no feasible solution.

Restart Criteria

For heuristic methods, when they either stall or reach a local minima various restarting (or diversification) methods are used. Most methods either gradually climb out of a hole or else fly to another location, or perform an expensive “supermove.” Some examples of climbing out of a hole are rolling back the annealing schedule, making certain moves tabu, tunneling methods involving either temporary or permanent penalty functions, or temporary penalties on the

weights of a neural network. One example of flying out of a hole is a random restart method. Random restart can work well for small and medium-sized problems, but as Boese et al. [1994] point out, there is a serious problem for larger problems. The local optima for large problems suffer from the phenomena of "central limit catastrophe." Local optima grow exponentially with the size of the problem, and the optima tend to be more of "average" quality. Boese et al. [1994] have a solution to this problem with a fascinating extension of random restart called adaptive multi-restart. They surmise that the location of local minima conform to a "big valley" with the global minima at the bottom of the valley. After some initial random restarts, they choose new starting locations based on the values of the previous global minima. One can see similarities between this method and a Benders decomposition restart method, where the "function evaluations" for the Benders decomposition are the values of the global optima.

The "supermove" method uses simple moves, such as swaps, as long as the function value is improving, and a more expensive move to get out of a local optima. For jobshop scheduling, one kind of proposed supermove involves trying two moves and choosing the best one. In the first move, the current time penalties, or some fraction of them, are added onto the setup costs and an assignment problem is solved. In the second move, the setup costs, or some constant fraction of them, are added to the current linear time penalties and the (NP-Hard yet quick) problem is solved.

Parallel Computing Issues

Gendron and Crainic [1994] give a survey of branch and bound methods. They note not just one but three types of parallelism in branch and bound algorithms. The three types are 1) parallel bounding operations but still serially building the branch and bound tree, 2) building the branch and bound tree in parallel, and 3) building several branch and bound trees in parallel. Gendron and Crainic [1994] state that parallelism of type 3 is easily applied to parallel computers. Using some of their classifications plus this author's own experience, one can extend these three types to other solution methods besides branch and bound. For example, one can solve one or all "threads" on one processor, but the other processors assist with costing and feasibility calculations. This parallelism of type 1 is especially amenable to shared memory processors and a small number of processors. An analog to type 2 parallelism for genetic methods is to have different "individuals" on various processors (remembering there will likely be many more individuals than processors), but having only one common population. Type 3 parallelism for genetic methods would be to have separate populations on different processors, with the possibility of interaction between them. Different populations could have different mutation rates, crossover operations, or other different characteristics.

Summary of Implementation Aspects Common to All Methods

Common Aspect	Possibilities
1. Approach	
Problem characteristics	Fixed no. batches? prize-collecting? makespan, timeliness, costs, production
Sequencing Requirements	No. of and multiplicity of jobs, operations, resources. Pipe headers? Optional resources?
Continuous Aspects	Strictly or heuristically separable from sequencing aspect
Types of deception	
Optimality Requirements	Optimal, near-optimal, Need good choices for all elements or only first few
Number of solutions	Find one best solution or many/all best/good solutions
Speed Requirements	How fast most of the time. How often can it be unacceptably slow?
Quality Requirement	Tightness of lower bound for comparison to solution
Formulation	Job or assignment based. Job-centric, operation-centric, or resource-centric
Domain decomposition	None, Static overlapping, Dynamic overlapping, non-overlapping
2. Initialization	
Time Required	Amount of time spent to get an initial solution and global lower bound
Simplification Tests	Entire problem and regions of the problem
Pre-conditioning	Setups costs and time. Pre-optimization, Dynamic program. precalculations
3. Evaluation	
Global cost calculations	Rigorous, Approximate, Use both, When to change
Feasibility evaluation	Ensuring feasibility intrinsically or through repair, Constraints, Penalties, Shifting penalty functions
Termination	
Stopping Criteria	Global lower bound, local optimum, stalling and cycling, out of time, User-choice, No feasible solution, Limit of restarts
Restart Method	None, Hill-climbing, Tabu, Tunneling, Random restart, GRASP, Adaptive multi-restart or other Benders decomposition type of methods
Parallel Computer Issues	
Intra-element granularity	Parallel computation for a particular element of the current proposed solution
Extra-element granularity	Parallel computation of different elements of the current proposed solution
Extra-solution granularity	Parallel computation of different proposed solution
Parallel communication	Learn from other processors prior to or else during processing of a solution

Communication Initiation	Polled or On-Demand

Table 3.13. Potential implementation aspects common to all methods.

Summary of Some Solution Techniques and Key Issues

A current unsolved challenge is the ability to look at a problem and reliably know, a priori, the best technique or combinations of techniques to use. In choosing an algorithm approach, key information includes:

Whether a rigorous optimal solution is required versus a "robust" optimum
Quality of the solution versus time to finish
What continuous aspects the problem has
If the problem is known to have one or a few local minima.
The expected roughness of the solution space
What algorithms have already had some success on problems similar to the one at hand.

There are a large number of important issues common to most algorithms, such as initialization procedure, problem formulation, calculating the exact or approximate cost function, and stopping criteria. Table 3.14 lists a few of the key issues specific to just a particular type of algorithm.

Approach	Key Issues
Exact and Heuristic Algorithms	
LP and NLP	How to avoid or escape local minima with NLP?
MILP and MINLP	Which combinatorial method(s) are used?
Enumeration	How is the bookkeeping done? Number of unique batches.
Branch and Bound and Beam Search	What are the bounds and how tight are they? Number of unique batches.
Dynamic Programming	What are the storage requirements? Number of unique batches.
Exact methods for special problems	How well does the problem map to an existing technique?
Heuristic Only Algorithms	
Genetic Methods	Deception avoidance. How to determine selection, crossover, mutation, adaptation? Many ways to formulate the problem.
Annealing Methods	How is the annealing schedule determined?
Expert Systems	Are candidate lists with more than one candidate tried?
Dispatching Rules	How is it decided which rule to use?
Artificial Neural Networks	What benefits are there to using this approach?
Tabu Search	Cycle avoidance. Absolute or relative position tabus
Ad-Hoc Algorithms	
Traveling Salesman Algorithms	How close is the problem to a traveling salesman problem?
Other Algorithms	How well does the problem map to an existing technique?

Table 3.14. Summary of scheduling solution techniques.

Unfortunately, one cannot simply look at a table such as Table 3.14 and know for certain which one algorithm or hybrid is best to use. A basic approach might not work well, but after

some modifications it could then work very well. One often needs to try multiple approaches until a modification of an algorithm is found that performs acceptably.

List of References

Boese, K.D., A.B. Kahng, and S. Muddu. A New Adaptive Multi-start Technique for Combinatorial Global Optimizations. *Operations Research Letters* vol.16 1994 p.101-113.

Dennis, J.E. Jr. and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* Prentice-Hall, Inc. 1983.

Gendron, B. and T.G. Crainic. Parallel Branch-and-Bound Algorithms : Survey and Synthesis. *Operations Research* vol.42 no.6 November-December 1994 p.1042-1062.

White, D.J. The Use of Specially Structured Models for Obtained Bounds in the Quadratic Assignment Problem. *Journal of the Operations Research Society* vol.45 no.4 1994 p.451-462.