(2) It has multiple resource types and multiple units of each resource type are required to execute an activity. Each resource type is a "shared" resource. If r_i^k units of resource type k are required by activity i then they must be allocated exclusively to activity i for i's entire duration. When activity i is completed, resources allocated to activity i are released to be used in other activities.

(3) The start time of each activity should be assigned to satisfy given precedence relations and resource constraints. Precedence relations are expressed in a standard activity-on-nodes (AON) project network. Resource constraints simply state that the number of units of resource type k in use at any time cannot exceed the overall availability of resoruce type k. This availability is assumed known and constant throughout the entire project duration.

(4) The effectiveness of the schedule is evaluated by various objectives, e. g., the completion time of the project, then, objective is to minimize project makespan.

The problem can be formulated in an intuitively simple and general fashion as follows:

Minimize t_n

subject to

$$t_i \geq \underset{j \in p_i}{\text{maximum}} \{t_j + d_j\} \quad \forall\, i \in N \qquad (1)$$

and

$$R^k \geq \sum_{i \in N_t} r_i^k \qquad \forall\, t \in [0,\ T-1] \text{ and } \forall\, k \in K \qquad (2)$$

where,

t_i = start time of activity i (decision variable), ≥ 0,

d_i = duration of activity i,

p_i = set of predecessor activities of activity i,

R^k = availability level of resource k,

N_t = set of activities in process in time interval (t, t+1),

r_i^k = number of units of resource k required by activity i,

$N = \{0, 1, 2, \cdots\cdots, n\}$: set of activities (with 0 and n the dummy activities "start" and "finish"),

K = set of resource (machine) types,

T = makespan of current schedule.

2. Disjunctive Graph Representation

Balas represents the job shop scheduling and the machine sequencing on a disjunctive graph [1], [2]. A disjunctive graph G is denoted

$G = (N, C, E)$,

Where N is the set of nodes, C the set of conjunctive arcs (precedence relations), and E the set of disjunctive arcs.

Figure 1. An example of disjunctive graph.

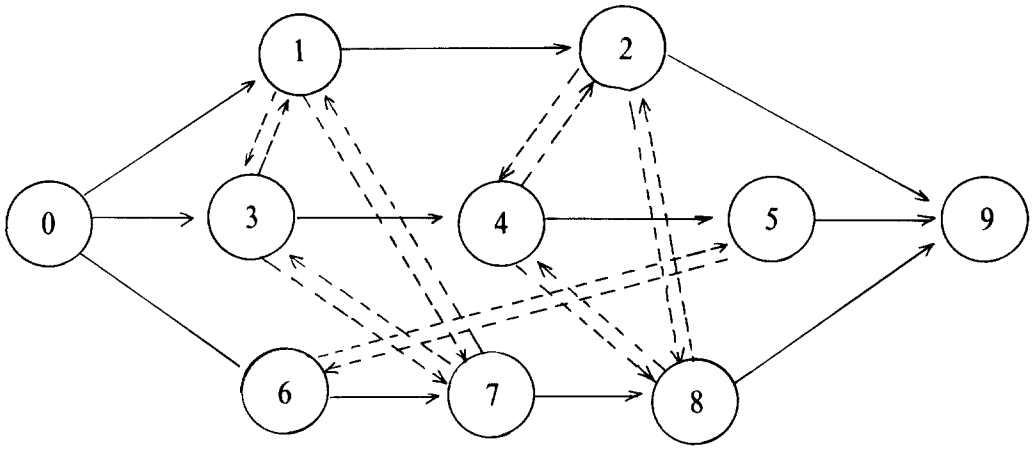


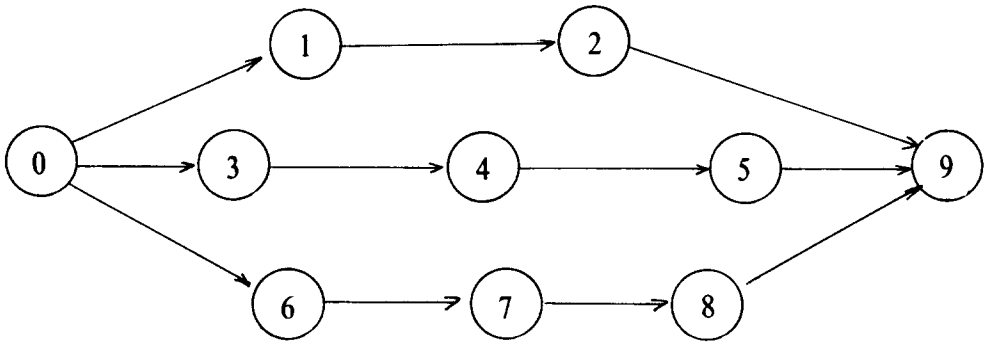
Figure 1 illustrates such a disjunctive graph for a job shop scheduling example with 3 jobs and 3 machines. If N^k denotes the operations pertaining to machine k, E^k the set of disjunctive arcs for machine k and an arc (i, j) the precedence relation of "from operation i to operation j", then $N=\cup(N^k : k \in K)=\{0, 1, 2, \cdots\cdots, 9\}$, $C=\{(0, 1), (1, 2), (2, 9), (0, 3), \cdots\cdots, (8, 9)\}$, and $E=\cup(E^k=\{(i, j), (j, i)\}, \forall i, j \in N^k$ and $\forall k \in K)$. Then the job shop scheduling or machine sequencing problem reduces to that of finding a minimaximal path in a disjunctive graph [1].

In the multi-resource constrained project scheduling, however, since $N^k \cap N^l \neq \phi$ and an operation may require multiple units of multiple resource (machine) types as shown in Figure 2, the cardinality of the set of disjunctive arcs E gets increased trememdously in Balas' disjunctive graph representation. Wherever the resource requirement of parallel activities exceeds the resource availability a disjunctive are must be introduced. Balas' disjunctive graph representation and approach of solving the one-machine problem is not appropriate.

3. The Resolution of Resource Conflicts by Disjunctive Arcs

All of the exact solution techniques [4], [5], [10], [11], [13], [14] developed in the resource constrained project scheduling are generally appropriate only for small-or moderate-size problems (e.g., up to 50 activities). This relative lack of success of optimization procedures in

Figure 2 A simple network example.



Act. No.	Res. Req	Duration	Startime
0	(0 0 0)	0	0
1	(3 2 4)	6	0
2	(6 2 3)	7	6
3	(4 5 3)	2	0
4	(1 3 3)	4	2
5	(1 2 5)	8	6
6	(4 1 4)	5	2
7	(3 2 5)	7	5
8	(1 5 2)	3	12
9	(0 0 0)	0	14

resource availabilities (10 10 10)

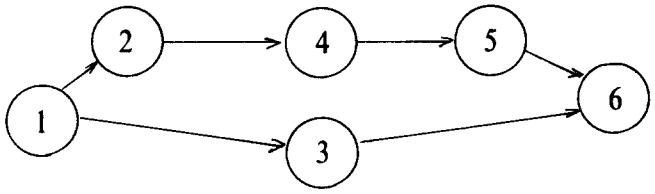
problems of realistic size motivated the development of heuristic solution methods which can generate reasonably good feasible solutions.

In previous heuristic methods which were "priority dispatching" procedures or variations, or even in some optimal solution methods, a set of activities are sorted and scheduled up to the resource capacities. This process can be illustrated in the simple example in Figure 3 taken from [13]. Figure 4 is its resulting branch and bound tree. Node 1 in the tree represents the only feasible scheduling decision at time zero : to place activity 1, the dummy start activity, in an active status. When activity 1 is completed, again at time zero, there are three possible feasible scheduling decisions.

The first is to schedule activity 2 by itself (node 2 in the tree);the second is to schedule activity 3 by itself (node 3);and the third is to schedule both activities 2 and 3 (node 4) since both taken together do not require more units of resource than are available. Although Figure 4 shows the entire branch and bound tree for this problem, each path leading to a terminal node (nodes 20 through 25) can be interpreted as a sequence of decisions dispatching activities.

Given the network structure of the above problem which satisfies precedence constraints, the scheduling problem can be narrowed down to the resolution of resource conflicts that are caused by concurrent processing of activities in the network. Because the amounts of available resources are not always sufficient to satisfy demands of concurrent activities, new sequencing

Figure 3. A simple example of a project scheduling problem.



Activity No.	Duration	Resource Req.	Start time
1	0	0	0
2	1	1	0
3	4	1	0
4	4	2	1
5	4	1	5
6	0	0	9

Resource availability=2 units

decisions are required whenever resource constraints are violated. Such sequencing decisions, which can be expressed in the network as the insertion of a new arc to the network, can lead to an increase in project duration beyond that determined by the original critical path calculation ignoring resource constraints. The objective is to make the required sequencing decisions so as to minimize the increase in project duration, subject to given resource and precedence constraints. Fulfillment of this objective would result in a minimum makespan schedule which satisfies both given precedence constraints and resource usage constraints.

A new approach in scheduling activities is to utilize the idea of a disjunctive constraint of Balas [2] to express alternative ways of resolving resource conflicts. A hypothetical schedule is constructed in which each activity starts at its earliest possible time with respect to original precedence constraints (as shown in the network of Figure 3). This schedule can be found by solving the mathematical formulation without constraint (2) in the section 1. This schedule is then examined for possible sets of activities whose concurrent execution causes resource violation. A set of activities N_t which together require more units than available of at least one resource at time interval (t, t+1) is called a violating set. For any such violating sets, all subsets which are minimal resource violating sets (MRV'S) are indentified and considered for the resolution of resource conflicts. A minimal violating set S is a violating set with no violating proper subsets. Then the

most early recognized MRVS is resolved by adding one of disjunctive arcs from disjunctive constraints

Figure 4. The branch and bound tree

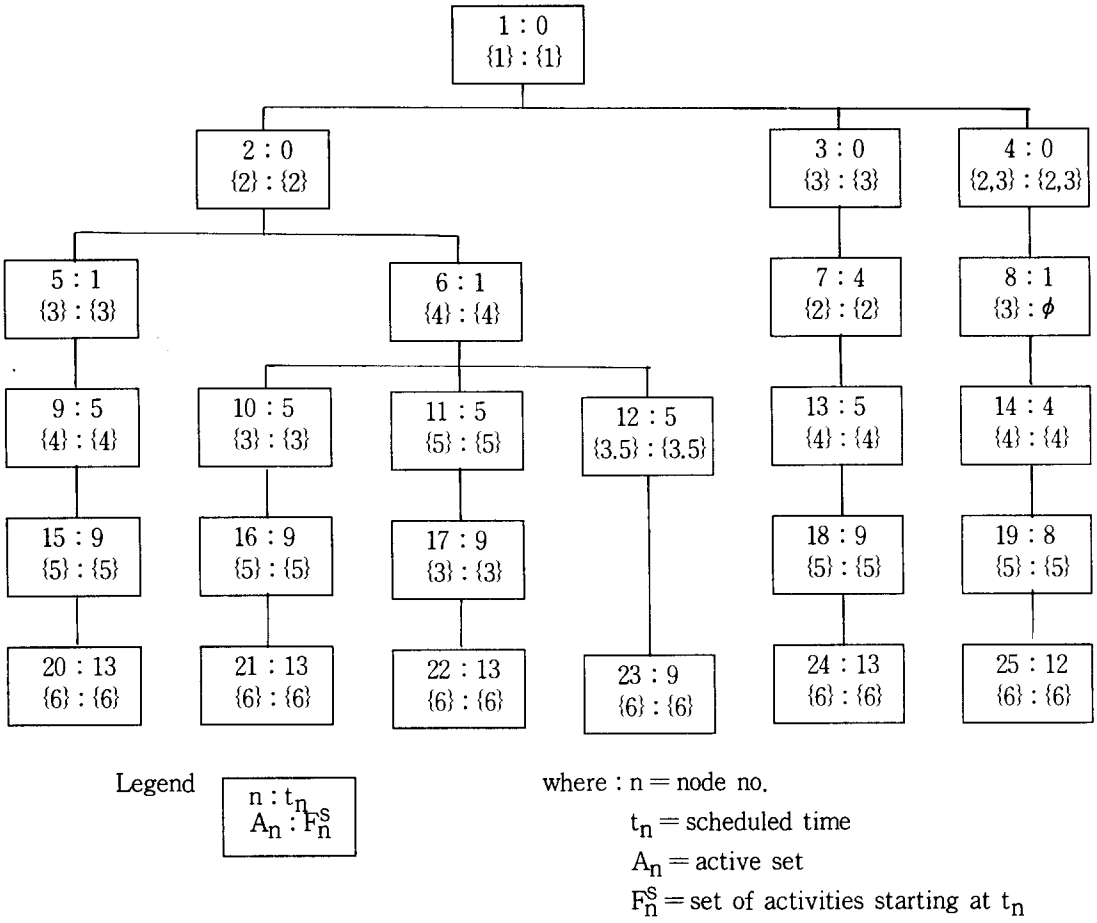


Figure 5. Minimal Resource Violating Set sat each time interval in the simple example.

time interval (t, t+1)	resource violating sets
0	NIL
1	((a4 a3))
2	((a4 a3))
3	((a4 a3))
4	NIL
5	NIL
6	NIL
7	NIL
8	NIL

$t_j - t_i \geq d_i \vee t_i - t_j \geq d_j, \forall i, j \in S, \forall S \subset N_t$

The MRVS's of the simple in Figure 3 are shown in Figure 5. MRVS of activity 4 and activity 3 generates disjunctive arcs (4, 3) and (3, 4). This resource violation is resolved by choosing a disjunctive arc which is guided by heuristic rules, e. g., minimize the resulting increase in project duration. With this heuristic, the disjunctive arc (4, 3) is selected to be inserted into the network since it does not increase the project duration while inserting (3, 4) would increase the project duration by 3 time units. A new network is determined. All activities are scheduled at their earliest times with respect to the network and this procedure is repeated until there are no more MRVS's. In this example, addition of the single disjunctive arc (4, 3) resolved all resource conflicts.

Even though the final solution by this method is the same as that found by a traditional priority dispatching procedure, the approaches used in sequencing conflicting activities are completely different. A priority dispatching procedure assembles a schedule by dispatching activities as time goes on. The effect of the sequencing procedure on the network cannot be readily observed. The new approach puts arcs into the precedence network for the resolution of resource conflicts. That is, while the traditional method is to schedule activities at a particular instant as long as resources are available and then to move on to the next appropriate scheduling instant, the new approach is to delay one of the conflicting activities by adding a new precedence (disjunctive) arc. Disjunctive constraints are easily recognized and can be remembered. This makes it possible to change the decision on which arc to add to the network. Thus, after adding arc (4, 3), it is possible to retract this arc and add (3, 4) instead. This option is used in the post-analysis phase explained below.

4. A New Algorithm

The algorithm has two phases([3], [9]) : (1) phase 1 (schedule generation phase) to generate a network whose early time schedule is found by successively repairing resource violations until no resource constraint violations remain (an "early time schedule" has each activity scheduled at its earliest possible time) and (2) phase 2 (limited search post-analysis phase) to use a limited search technique which examines the critical path of this network and attempts to find local improvements in the schedule. In both phase 1 and phase 2 the new heuristic algorithm always make choices with a certain objective in mind. In the following explanation we assume that the objective is to minimize the makespan of a project.

Phase 1, the schedule generation phase, is executed without backtracking. The algorithm analyzes the current set of expressed precedence constraints and recognizes resource violations.

It then finds a resource violating set of activities by examining the "early time schedule" implied by the current precedence constraints.

If a minimal resource violating set S is found with activities $\{a_i, a_j, \cdots\cdots, a_k\}$, the algorithm then "decides" which disjunctive arc, e. g., in the form of $a_i \rightarrow a_j$, should be added to the network. Various decision rules are possible. Thus the behavior of the schedule generation phase of the algorithm is determined by its choice of resource violating set (if there are two or more) and its decision rule for adding an arc to guarantee that not all members of the chosen resource violating set can be executed concurrently. As currently implemented, the new algorithm scans the early time schedule starting at time 0 and finds the first time interval (t, t+1) with one or more MRVS's.

The choice of an appropriate disjunctive arc among $(a_i \rightarrow a_j)$'s generated from S is decided by the following lexicographic rule :

(i) select $a_i \rightarrow a_j$ such that
$\underset{a_i \rightarrow a_j}{\text{miminum}}\ EFT_i + (T - LST_j)$,

(ii) select $a_i \rightarrow a_j$ among disjunctive arcs still consideration such that
$\underset{a_i \rightarrow a_j}{\text{minimum}}\ EST_i$,

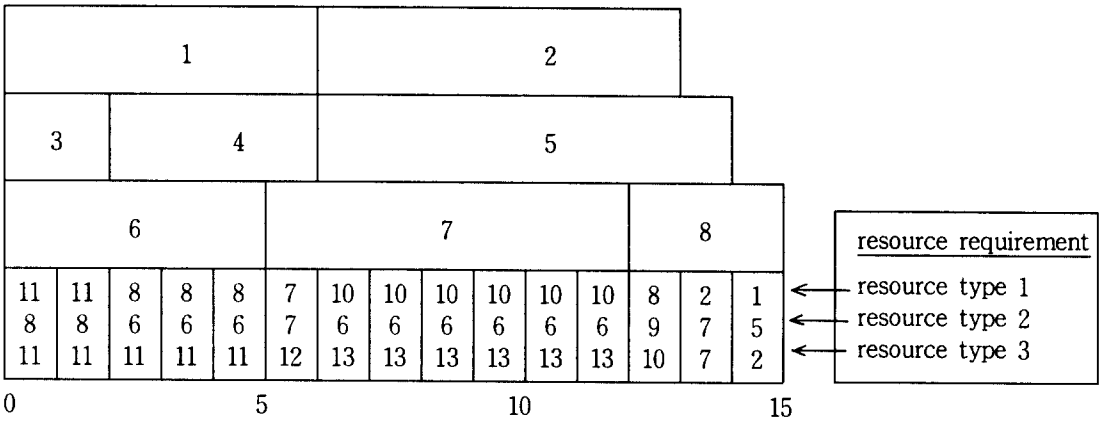
where EFT_i denotes the earliest finish time of activity i, LST_j the latest start time of activity j and EST_i the earliest start time of activity i.

When all resource violations are resolved the first feasible solution is found in early time schedule. The first phase can be illustrated by the example of Figure 2. The starting network and its associated early time schedule (ignoring resource constraints) is shown in Figure 2 and its Gantt chart in Figure 6. The first MRVS can be recognized in time interval (0, 1) as the set of activities {a1, a3, a6}. Having found an MRVS, {a1, a3, a6}, one of a1→a3, a1→a6, a3→a1, a3→a6, a6→a1, and a6→a3 can be selected. By adding an arc a3→a1, the critical path length remains unchanged.

In the revised network including arc a3→a1, the first MRVS can be found in time interval (2, 3) as the set {a1, a4, a6}. Adding arc a6→a4 will cause the least increase in critical path length. The process is continued until a network with no resource violating sets is finally reached. When the first feasible schedule is found following arcs have been added : a3→a1, a6→a4, a1→a7, a4→a2, and a7→a2, and the resulting critical path length is 22. This solution will be used as the root node in the second phase.

In phase 1, arcs were added to the original precedence network in a somewhat myopic fashion (to correct a particular resource violation). It might well be the case that one or more arcs added through this process are no longer needed. Thus, whenever we have just found a feasible solution, we execute a simple procedure, Clean_Up_Network, to remove any added arcs which have become redundant. Removing such redundant arcs has no impact on the makespan of the

Figure 6. Gantt chart for a project scheduling problem.

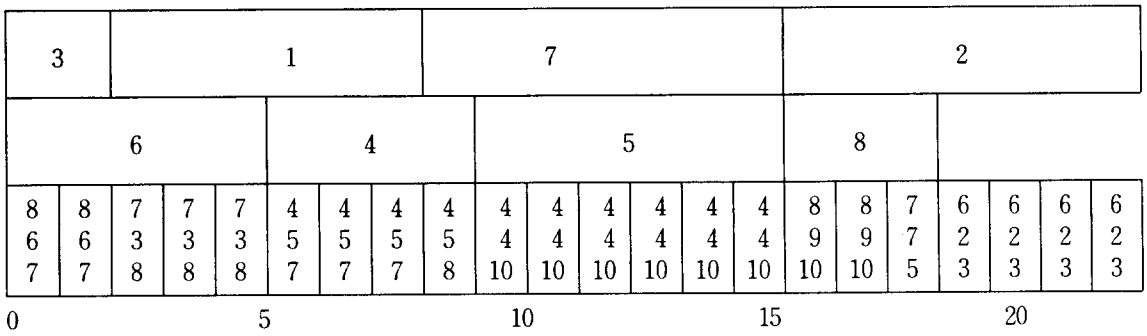


current feasible solution, but can have a beneficial impact later as the limited search phase 2 proceeds : less computation time and possible better solution due to less-constrained network.

Clean_Up_Network works very simply. Added arcs are examined in turn. If an added arc can be removed from the network without destroying feasibility, then this is done. Otherwise, the added arc is left in the network. Thus, after executing Clean_Up_Network, the remaining set of added arcs is a subset of the set of added arcs associated with the current feasible schedule when it was first descovered. Among newly added arcs in the example, the added arc a4→a2 turns out redundant and is removed. The solution, now, is updated as a3→a1, a6→a4, a1→a7, and a7→a2. Gantt chart of the first feasible schedule is shown in Figure 7.

In the second phase we exanime, in turn, the effect of removing each of the added arcs on the critical path and using the resulting network (which will typically now have at least one resource violation) as a starting point for generating a new feasible solution. In each case, the addition of every alternative arc for correcting the newly revealed resource violation is explored. After putting in one such arc, the phase 1 heuristic is then used to find a new solution. These efforts are abandoned as soon as we can demonstrate that any new solution found by this heuristic will not improve on the incumbent best feasible solution. If an improved solution is found, the process of phase 2 is repeated starting with that solution.

Figure 7. Gantt chart of the first feasible solution.



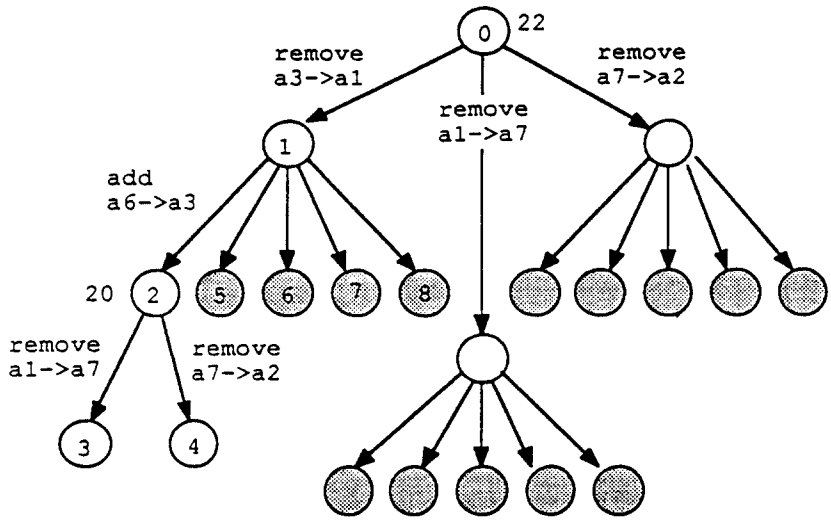
Thus the entire algorithm can be described by the procedure Search_For_Network with pseudocode presented below. Incumbent_Solution is a global variable, initially nil. Makespan (Incumbent_Solution) returns a large positive number if Incumbent_Solution is nil, otherwise it returns the actual makespan. Added_Critical_Arcs (Network) returns a list of all arcs which are on the critical path of Network and which were not in the original precedence network. Add_Arc (Arc, Network) and Remove_Arc (Arc, Network) each return a new Network with the given Arc either added or removed. The critical procedure which affects performance is Violating_Sets (Network) which returns resource violating sets for a Network known to contain at least one. If Violating_Sets (Network) is nil, Feasible (Network) becomes true. Alternative_Arcs (Arc, VS) generates possible disjunctive arcs one of which is to be added to the network to "break up" the violating set VS, and Arc_To Add (VS, Network) returns an appropriate arc to be added to the network. Network, VS, Alternative and Arc are local variables.

```
PROCEDURE Search_For_Network (Network)
  IF Makespan (Network) < Makespan (Incumbent_Solution) THEN
    IF Feasible (Network) THEN
      Clean_Up_Network (Network) ;
      Incumbent_Solution ← Network ;
      FOR Arc in Added_Critical_Arcs (Network) DO
        Network ← Remove_Arc (Arc, Network) ;
        VS ← Violating_Sets (Network) ;
        FOR Alternative in Alternative_Arcs (Arc, VS) DO
          Search_For_Network (Add_Arc (Alternative, Network)) ;
        ENDFOR ;
        Network ← Add_Arc (Arc, Network)
      ENDFOR
```

```
    ELSE
      VS ← Violating_Sets (Network) ;
      Arc ← Arc_To_Add (VS, Network) ;
      Search_For_Network (Add_Arc (Arc, Network))
    ENDIF
  ENDIF
ENDPROCEDURE.
```

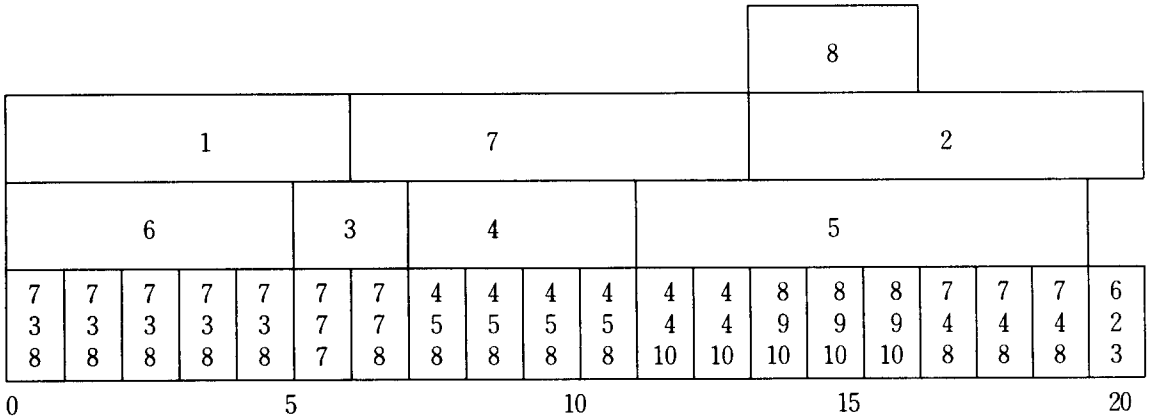
Returning to the numerical example, the feasible solution generated by the first phase included the following additional arcs : a3→a1, a6→a4, a1→a7, a7→a2. The critical path is analyzed and found to contain the following added arcs : a3→a1, a1→a7, a7→a2. Each of these arcs in the critical path is removed in turn from the network to investigate the building of alternative feasible solutions.

Figure 8. Search tree in post-analysis.



In Figure 8, the first feasible solution is represented by node 0. The first arc in the critical path, a3→a1 is replaced by the alternative arc a6→a3 and an improved solution (node 2) with makespan 20 is found immediately. The set of added arcs in this solution arcs is {a6→a3, a6→a4, a1→a7, a7→a2}. Clean_Up_Network allows the removal of an arc a6→a4 from the improved solution. The phase 2 process is repeated with the improved solution : a6→a3, a1→a7, a7→a2 (only a1→a7 and a7→a2 on the critical path). Further attempts to improve the solution prove fruitless (all other node in the search tree in Figure 8 are pruned by the incumbent makespan 20). Gantt charts of the first feasible solution and the final solution are shown in Figure 9.

Figure 9. Gantt chart of the final solution.



5. Results of Computational Experiments

The experiment consisted of successively solving 110 single project, 56 multiproject and 40 job shop scheduling problems. Those 110 problems, used in [12], represent an accumulation of all single project scheduling problems existing in the literature today.

For the 56 multiproject scheduling problems, eight mock projects were formulated by modifying those in [7]. The ratio of number of arcs to number of activities in these problems varied between 1.0 and 2.0. Activity durations varied between 1 and 9 time units and were generated using the Beta distribution with parameters a=b=0.5 and range [1,9]. Observations were rounded to the nearest integer.

Each activity is assumed to require 0, 1, 2, or 3 units of each of three scarce resource types. Such resource requirements were generated randomly from the discrete uniform distribution on [0, 3]. These resource requirements were assigned before the experiment was begun and were not changed during the experiment. Resource availability of six units of each resource at each time instant was assumed. All combinations of three mock projects in parallel were constructed to make 56 multiproject problems. The earliest start time of each project in the project set was generated randomly from the discrete uniform distribution on [0, 20].

The 40 job shop scheduling problems are those adapted from [1]. In these problems, each job is to be processed on every machine, the sequence of machine for each job is random, and the processing times are randomly drawn integers from the inteval [5,99].

Given the set of 110 test problems, we first selected 83 problems identical to those in [6]. For the 83 problems chosen initially, Table 1 gives a comparison of the performance of the new heu-

ristic solution with known optima. The new algorithm gave an average increase of 5.7% above the optimum makespan for the first feasible solution found. The limited search second phase of the new algorithm reduced this average increase to 3.1% above the optimum. The new heuristic algorithm outperformed all other heuristic rules in [6].

Based on the survey of previous research, six heuristic rules were selected for the multiproject scheduling problems. For the detailed explanations of each heuristic rule see [9]. Each of Table 2 through 4 shows a performance comparison of the new algorithm with a priority dispatching procedure in each of the heuristic rules. In 56 multiproject problems, three rules, LET, MINSLK, and RSM*[1], constitute a group which produced generally better results than the other three. When comparing the performance of the new algorithm to that of a priority dispatching procedure, it was found that the performance of LET in priority dispatching was almost equivalent to that of LFT, MINSLK, and RSM* in the new heuristic algorithm even though RSM* in the new algorithm performed slightly better. However, given similar performance in the total delay criterion, RSM* in the new algorithm performed best in minimizing total resource idle time (which can be interpreted as minimizing makespan of the project set). For example, RSM* in the new algorithm produced 366 time units total resource idle time on the average while the next best idle time was 430 time units by LFT in the priority dispatching procedure.

Table 1. Comparison of heuristic and optimal solutions for 83 problems.

	Phase 1		Phase 2		[6]
	%diff	sol'n	%diff	sol'n	%diff
mean	5.7	36.2	3.1	35.3	5.6
S.D.	4.6	7.8	3.2	7.5	6.1

Table 2. Comparisons in total delay of multi project sheduling problems.

	the new algorithm						priority dispatching procedure				
	grd	lft	minslk	rsm*	sjf	ran	grd	lft	minslk	rsm*	sjf
mean	62.1	46.0	45.2	44.2	56.8	59.5	70.5	45,2	50.1	66.7	70.6

1 : the heuristic rule used in this research : a variation of RSM in [3].

Table 3. Comparisons in weighted total delay of multi project scheduling problems.

	the new algorithm						priority dispatching procedure				
	grd	lft	minslk	rsm*	sjf	ran	grd	lft	minslk	rsm*	sjf
mean	3278	2574	2533	2536	3073	3282	3642	2528	2865	3465	3552

Table 4. Comparisons in total resource idle time of multi project scheduling problems

	out heuristic algorithm						priority dispatching procedure				
	grd	lft	minslk	rsm*	sjf	ran	grd	lft	minslk	rsm*	sjf
mean	435	495	478	366	483	479	395	430	464	404	526

Table 5. The number of best solutions produced by each heuristic rule.

	the new algorithm					priority dispatching procedure				
	grd	lft	minslk	rsm*	sjf	grd	lft	minslk	rsm*	sjf
total delay	0	10	12	23	2	0	16	7	0	0
weighted total delay	0	8	11	20	3	0	10	4	0	0
idle time	3	0	2	29	2	12	2	4	9	0

A different aspect of performance comparison is shown in Table 5 which gives data in terms of the number of times each rule produced the best solution (including ties) out of 56 test problems. In each performance criterion RMS* in the new algorithm found a shortest-total-delay schedule most often, followed by LFT in priority dispatching procedure. GRD and SJF rules performed worst in general. While these results are hardly surprising, they support earlier observations and illustrate that there seems to be a strong coupling between the RSM* rule and the new algorithm.

In overall performance, in terms of the number of best solution (excluding ties) produced by

each method, the new method produced the least-total-delay solution in 35 problems and the priority dispatching procedure generated the best schedule in 13 problems.

On the 40 job shop scheduling problems the new algorithm outperformed the priority dispatching procedure as shown in Table 6 : 4.7% (on the average) improvement in the solution, even though Adame, Balas and Zawack had produced better solutions in [1] : 8.5% improvement. However, one of the advantages of our new algorithm is that it can be applied to both project scheduling and job shop scheduling while the method in [1] is applicable only to job shop problems. It can be suggested that the new algorithm be refined to be able to take advantage of the characteristics of the job shop problem to be more effective.

6. Conclusions

Scheduling is a challenging but potentially very frustrating decision-making problem. Even though the problem is easy to state and to visualize, the substantial literature does not include any sophisticated and clever way of finding an exact optimal solution to the problem of project and job shop scheduling. This is not surprising since the problem is known to be NP-complete [8]. Therefore heuristic solution schemes have been utilized to get reasonably good solutions in short computation times. Here we developed a new approach to finding a heuristic solution to the project scheduling. It works in job shop scheduling as well.

In addition to the advantage of simplicity and effectiveness, the new heuristic method has several unique characteristics :

(1) The new algorithm utilizes a philosophically different approach in generating a feasible schedule : namely, it adds one new precedence arc at a time to repair a resource violation. Particularly, since the representation scheme of this approach is compatible with those of AI automated planning systems such as NONLIN [15], DEVISER [16], etc. this philosophical approach can be implemented into those systems.

(2) The new algorithm employs an inexpensive limited search technique in the second phase to seek an improvement in the first found feasible solution. Direct comparison of computational times with those of other approaches for the same problems is difficult since prior studies, whose programming codes are not available, used mainframe computers and languages such as Fortran while computations reported in this paper were performed on an Apple Machintosh SE using a version of Common Lisp.

(3) The new algorithm works better as resource availabilities get larger so that fewer violating sets exist in the network while the existing algorithms (including exact algorithms) do worse. That is, existing algorithms which utilize the 'schedulable subsets' at each iteration of

Table 6. Performance comparison in job shop scheduling problems

problems	priority dispatching	phase 1		phase 2		Adams et al.	
		makespan	%	makespan	%	makespan	%
1	679	679	0.0	666	−1.9	666	−1.9
2	792	838	5.8	724	−8.6	669	−15.5
3	673	735	9.2	673	0.0	605	−10.1
4	670	696	3.9	639	−4.6	593	−11.5
5	594	612	3.0	606	2.0	593	−0.2
6	927	926	−0.1	926	−0.1	926	−0.1
7	947	1014	7.1	890	−6.0	890	−6.0
8	880	896	1.8	874	−0.7	863	−1.9
9	952	968	1.7	951	−0.1	951	−0.1
10	959	958	−0.1	958	−0.1	959	0.0
11	1223	1231	0.7	1222	−0.1	1222	−0.1
12	1041	1039	−0.2	1039	−0.2	1039	−0.2
13	1151	1174	2.0	1150	−0.1	1150	−0.1
14	1293	1292	−0.1	1292	−0.1	1292	−0.1
15	1320	1268	−3.9	1207	−8.6	1207	−8.6
16	1036	1101	6.3	1005	−3.0	978	−5.6
17	857	826	−3.6	814	−5.0	787	−8.2
18	897	963	7.4	924	3.0	859	−4.2
19	926	954	3.0	917	−1.0	860	−7.1
20	1001	1046	4.5	986	−1.5	914	−8.7
21	1208	1193	−1.2	1153	−4.6	1084	−10.3
22	1085	1060	−2.3	1004	−7.5	944	−13.0
23	1163	1139	−2.1	1102	−5.2	1032	−11.3
24	1142	1153	1.0	1007	−11.8	976	−14.5
25	1259	1154	−8.3	1058	−16.0	1017	−19.2
26	1373	1373	0.0	1288	−6.2	1224	−10.9
27	1472	1517	3.1	1384	−6.0	1291	−12.3
28	1475	1449	−1.8	1411	−4.3	1250	−15.3
29	1539	1360	−11.6	1290	−16.2	1239	−19.5
30	1604	1460	−9.0	1424	−11.2	1355	−15.5
31	1935	1906	−1.5	1846	−4.6	1784	−7.8
32	1969	1871	−5.0	1850	−6.0	1850	−6.0
33	1871	1860	−0.6	1754	−6.3	1719	−8.1
34	1926	1833	−4.8	1749	−9.2	1721	−10.6
35	2097	2057	−1.9	1889	−9.9	1888	−10.0
36	1517	1415	−6.7	1404	−7.4	1305	−14.0
37	1670	1590	−4.8	1507	−9.8	1423	−14.8
38	1405	1459	3.8	1364	−2.9	1255	−10.7
39	1436	1571	9.4	1435	−0.1	1273	−11.4
40	1477	1437	−2.7	1395	−5.6	1269	−14.1
mean			0.0		−4.7		−8.5

execution generate more choices at each decision point as resource constraints get less tight. An increase in the number of possible schedulable subsets (higher branching factor in the search tree) due to greater resource availability makes the search tree bushier and requires that the algorithm do more work. However, the new algorithm requires less work because there are fewer resource violating sets when resource availability is increased.

Finally, some possible relaxations in the assumptions can be considered. Often, a given problem type or a practical real problem does not satisfy all of the assumptions built in to the design of the new algorithm. Two possible relaxations and recommendations for modifications are provided below.

(1) Relaxation of constant resource usage assumption : This relaxation can be achieved by splitting an activity into a sequence of activities each of which employ a constant resource usage. However, when an activity is split into a sequence of activities in this manner, the sequence is restricted to be continuously executed since the original activity is required to be processed without preemption.

(2) Relaxation of constant resource availability assumption : Where the quantity of resources available over a schedule duration varies (caused by such factors as absenteeism, vacations, machine check−up schedule, etc), the simple way to accomplish this would be to use a dummy activity. Whenever a resource availability goes down from its highest availability level for a certain duration, such a dummy activity of that duration is introduced that requires a level of resource consumption equal to the difference between highest and actual availability level. This relaxation also requires some additional modelling. If the availability of resource r is dropped from its maximum level during the fixed time interval [t, t+d] then the dummy activity representing this decrease must be guaranteed to start exactly at time t and finish exactly at time t+d.

Thus, there appear to be significant opportunities for future research on scheduling problems. This and prior investigations have revealed that exact solution approaches are not promising due to the disjunctive nature of resource constraint. However, the new approach to developing heuristic methods used in this paper might well be adapted to other contexts with relaxed assumptions.

REFERENCES

1. J. Adams, E. Balas and D. Zawack, "The Shifting Bottleneck Procedure for Job Shop Scheduling", Management Science, Vol. 34, No. 3, pp. 391-401, 1988.
2. E. Balas, "Machine Sequencing Via Disjunctive Graphs : An Implicit Enumeration Algorithm", Operations Research, Vol. 17, No. 6, pp. 941-957, 1969.
3. C.E. Bell and J. Han, "A New Heuristic Solution Method in Resource-Constrained Project Scheduling", Naval Research Logistics, Vol. 38, pp. 315-331, 1991.
4. J.D. Brand, W.L. Meyer, and L.R. Shaffer, "The Resource Scheduling Problem in Construction", Civil Engineering Studies, Report No.5, Department of Civil Engineering, University of Illinois, Urbana, IL, 1964.
5. E.W. Davis and G.E. Heidorn, "An Algorithm for Optimal Project Scheduling Under Multiple Resource Constraints", Management Science, Vol. 17, No. 12, pp. B803-B816, 1971.
6. E.W. Davis and J.H. Patterson, "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling", Management Science, Vol. 21, No. 8, pp. 944-955, 1975.
7. L.G. Fendley, "Toward the Development of a Complete Multiproject Scheduling System", Journal of Industrial Engineering, Vol. 19, No. 10, pp. 505-515, 1968.
8. M.R. Garey and D.S. Johnson, Computers and Intractability : a Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
9. J. Han, "A New Heuristic Algorithm for Resource-Constrained Project Scheduling", Unpublished Ph. D. Thesis, University of Iowa, Iowa City, IA, 1988.
10. T.J.R. Johnson, "An Algorithm for the Resource-Constrained Project Scheduling Problem", Unpublished Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1967.
11. J.H. Patterson and G. Roth, "Scheduling a Project under Multiple Resource Constraints : A Zero-One Programming Approach", AIIE Transactions, Vol. 8, No. 3, pp. 449-456, 1976.
12. J.H. Patterson, "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem", Management Science, Vol. 30, No. 7, pp. 854-867, 1984.
13. J.P. Stinson, E.W. Davis and B.M. Khumawala, "Multiple Resource-Constrained Scheduling Using Branch and Bound", AIIE Transactions, Vol. 10, No. 3, pp. 252-259, 1978.
14. F.B. Talbot and J.H. Patterson, "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems", Management Science, Vol. 24, No. 11, pp. 1163-1174, 1978.

15. A. Tate, "Project Planning using a Hierarchical Nonlinear Planner", Department of Artificial Intelligence Report 25, University of Edinburgh, UK, 1976.
16. S.A. Vere, "Planning in Time : Windows and Durations for Activities and Goals", IEEE Transactions on Pattern Analysis and Machine Intelligince, PAMI-5, No. 3, pp. 246-267, 1983.