International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: https://www.tandfonline.com/loi/tprs20

Balancing two-sided assembly lines: a case study

J. J. BARTHOLDI

To cite this article: J. J. BARTHOLDI (1993) Balancing two-sided assembly lines: a case study, International Journal of Production Research, 31:10, 2447-2461, DOI: 10.1080/00207549308956868

To link to this article: https://doi.org/10.1080/00207549308956868

INT. J. PROD. RES., 1993, VOL. 31, NO. 10, 2447–2461

Balancing two-sided assembly lines: a case study

J. J. BARTHOLDI

We describe the design and use of a computer program to balance two-sided assembly lines. The program embodies a balancing algorithm that emphasizes speed over accuracy for the interactive, rapid refinement of solutions. We argue that this is a more useful approach to balancing real assembly lines than the typical optimality-seeking techniques. We also discuss some theoretical properties of two-sided lines; and we include the data for a real assembly line, which to our knowledge is the first to appear in the open literature in over 30 years.

1. Introduction

A manufacturer of small utility vehicles rebalances its several assembly lines as often as once each week to better match production rate to market demand that is both seasonal and 'lumpy'. Each line produces a single model of vehicle. The assembly at each station is done by a worker who is either a permanent employee of the manufacturer or else a temporary employee hired from a pool of contract labour. The company wants to minimize the number of stations (workers) required to produce at any given rate. Adjustments to the number of workers are made by hiring the appropriate number of temporaries.

Each line has two sides, left and right, and at each position there is a pair of stations directly opposite each other, as shown in Fig. 1, so that each vehicle is worked on by two people simultaneously. Some tasks can be assigned only to one side of the line (e.g. 'mount the left wheel'); some can be assigned to either side of the line (e.g. 'install the hood ornament'); and some tasks must be assigned to both sides of the line simultaneously, so that the pair of workers on opposite sides can collaborate (e.g. 'install the rear seat'). Each of the several products consists of 80–300 tasks. (The data for one of the products appears in an appendix).

The manufacturer requested software to enable an engineer to rebalance a line quickly. It was especially important that the program be fast, convenient, and interactive. This was important not only for reconfiguring the lines, but also for

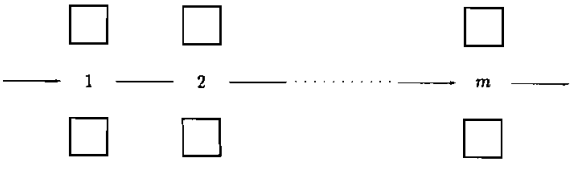

Figure 1. Structure of a two-sided assembly line.

Revision received December 1992.

School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA.

testing the effects of design changes on the manufacturing process. For example, an engineer might consider redesigning a fender, thereby changing some of the tasks associated with installing it; then he could evaluate the effects on labour requirements by using the program to rebalance the line with the hypothetical changes.

There is a large literature on assembly line balancing. However, most of it concerns algorithms to balance an idealized one-sided line and it places a narrow emphasis on small distinctions in speed and accuracy. Our paper complements this literature in several ways. First, we discuss two-sided lines, which are widely used but apparently heretofore unmentioned in academic literature. Also, we emphasize the complications faced in balancing real lines and suggest a richer set of criteria on which to judge line-balancing algorithms. We also briefly discuss how our choice of line-balancing algorithm has been embodied in a software architecture that supports and enhances its useful features. Finally, we include the data for a real assembly line, which to our knowledge is the first to appear in the open literature in over 30 years.

2. Balancing a two-sided line

On a two-sided line the workers at each pair of opposite stations work in parallel on different tasks but on the same individual item. Note that this is different from the workers collaborating on the same task and same item; it is also different from the workers performing the same task, but on different items, as in Pinto *et al.* (1981). It is practical for two people to work together on the same item only when they will not interfere with each other; thus two-sided lines are more likely to be practical for large products like vehicles than for small ones like electric drills.

A two-sided line offers several advantages over a one-sided line. On a two-sided line some task times might be shorter since the worker can avoid setup times in which he repositions himself for tasks like mounting a wheel on the other side of the vehicle. Also a two-sided line can be more space-efficient since the line can be shorter in length than a one-sided line. (In fact, as we show shortly, a two-sided line can require fewer stations than a one-sided line.) A shorter line can reduce material handling costs since it mitigates the need for workers to manoeuvre tools, parts, or the item. In addition, there might be savings when workers at a pair of stations can share tools or fixtures, such as electrical or air outlets.

Whether a two-sided line in fact delivers advantages over a one-sided line depends on the precedence constraints among tasks. As an extreme example, consider balancing a line in which each task requires 1 time unit and can be assigned to either side of the line, and where the precedences form a 'chain' in which task A must immediately precede task B, which must precede task C, and so on. For such a set of tasks, any two-sided line balance must have at least one of each pair of opposite stations completely idle, since no two tasks can be in process at the same time at opposite stations. The difficulty is that the precedence constraints are 'too narrow' to permit two stations to work in parallel, so that a two-sided line offers no advantages over a one-sided line in this example.

On the other hand a two-sided line can require fewer stations than a one-sided line, as shown by the example in Fig. 2. For a cycle time of 10, a two-sided line requires two stations (one with tasks A, C, and the other with tasks B, D), while a one-sided line requires three stations (one with tasks A, B, one with task C, and one with task D). This illustrates the sort of precedence for which two-sided lines seem well-suited: there is mirror image symmetry in tasks and in precedence; that is, every task has a mirror image task, and if task i must precede task j, then both task i and

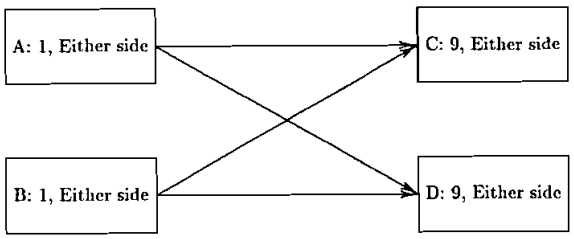

Figure 2. For this set of tasks and a cycle time of 10, a two-sided line requires fewer stations than a one-sided line.

its mirror image must precede the mirror image of task j, then both task i and its mirror image must precede the mirror image of task j.

The preceding examples show that one-sided and two-sided lines can require different numbers of workstations to produce the same product at the same rate. The following theorem shows that a two-sided line never requires more stations than a one-sided line, which seems intuitive—but, surprisingly a two-sided line can require as few as two-thirds as many stations as a one-sided line. This means that if all stations are the same physical size then a two-sided line can be as little as one-third the length of a one-sided line for the same product. Alternatively, with the same number of stations, a two-sided line can achieve a production rate up to 3/2 times that of a one-sided line.

Theorem 1. Consider a set of tasks for which each task can be assigned to either side of the line. For a fixed cycle time let z_2 be the minimum number of stations required in a two-sided assembly line, and let z_1 be the minimum number required in a one-sided line. Then $(2/3)z_1 \leq z_2 \leq z_1$, and these bounds are tight.

Proof. It is obvious that $z_2 \leq z_1$ since a one-sided line can be viewed as a two-sided line with some stations unused.

To shows that $(2/3)z_1 \leq z_2$, consider any two-sided line, and within it, any opposite pair of stations. It is possible to reassign the tasks at these two stations to no more than three consecutive stations devoted solely to these tasks on a one-sided line; for if four or more stations were required to contain these tasks, then since the total work content at each consecutive pair of stations on the one-sided line must exceed the cycle time, there must have been more than two stations worth of work. This contradicts the assumption that these tasks were feasibly assigned to a pair of opposite stations on a two-sided line. Since the tasks at each pair of stations on any two-sided line can be reassigned to no more than three stations on a one-sided line, $z_1 \leq (3/2)z_2$.

That the bounds are tight is established by the examples preceding the theorem.

By extending the argument in a straightforward manner, one can show the following bounds. Let z_k be the fewest workers required on a line in which there are k workers at each position; then $(kz_1)/(2k-1) \leq z_k \leq z_1$, and this bound is achievable. Thus for large number of workers at each position, as few as one-half the workers might be required compared to a one-sided line.

To balance a two-sided line is a difficult problem. Of course it is formally difficult in the sense of being *NP*-complete (Garey and Johnson 1979) since the same is true of balancing a one-sided line. It seems even more challenging to produce good

balances for two-sided lines since there is an additional level of complexity. As for a one-sided line, there is the difficult problem of assigning tasks to positions along the line (that is, stations on a one-sided line, or pairs of stations on a two-sided line). But for a two-sided line, even if the assignment of tasks to pairs of stations is given, it can still be hard to determine exactly how to schedule the tasks on any pair of stations since tasks at opposite sides of the line can interfere with each other through precedence constraints.

A simple example of interference is given in Fig. 3, the tasks of which are to be assigned to stations with cycle time of 10 units. Assume that task A is assigned to the first station on the left. Then that station must perform task A and then task B, for a total work load of 10 time units. On the other hand, the first station on the right must remain idle for 9 time units (until task A has been completed), and then it must perform task C, for a total work load of 1 time unit. Thus, in a two-sided line, some stations might be forced to remain idle, awaiting completion of a task by the opposite station. Note that this idle time occurs in the midst of the work schedule for the station. Consequently, during a cycle on a two-sided line, a station can be intermittently busy and idle. (Note that this interference can occur even when the tasks and precedence are symmetrical, since we break the symmetry when assigning tasks that can go to either side of the line.)

3. General structure of the program

The computer program consists of two main functional units, the product/task editor and the line/station editor. The product editor manages the work standards of a product, and the line editor manages the layout of the assembly line. Both editors support the incremental building and adjusting of balanced assembly lines. The user can toggle back and forth between the two editors, as suggested by Fig. 4.

The two editors are 'object-based' in that they allow the user to interact at a high conceptual level with products and their tasks and with assembly lines and their workstations (rather than only with the text describing them). For example, any such object can be selected, displayed, printed, modified, or deleted.

The product editor provides typical database capabilities. For example, the user can pop open a window and ask the program to search for tasks whose description includes the phrase 'right fender' or those which require a certain part. Additionally the user can view the tasks as nodes in a network of precedence constraints, and can move along the arcs of that network. For example, the user can jump to any of the immediate predecessors or successors of a task. Also, at any time the user can edit any of the data associated with a task, which includes the following:

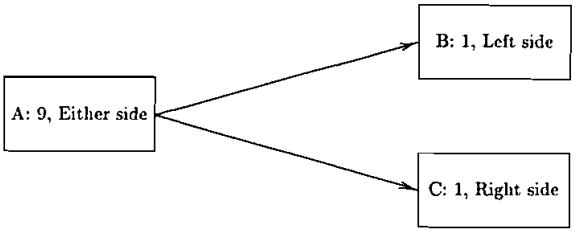

Figure 3. When this set of tasks is assigned to a two-sided line with a cycle time of 10, one station will have forced idle time in the midst of its schedule.

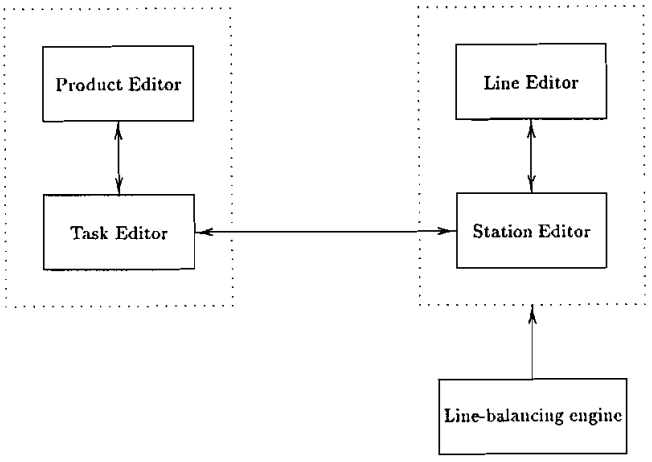

Figure 4. General structure of the program.

- a detailed text description of the task (which can be arbitrarily long);
- the time required to complete the task;
- the side of the line to which the task must be assigned (left, right, either, or both sides);
- a list of tasks that are immediate predecessors;
- a list of tasks that are immediate successors;
- a list of all parts required for the task (name, stock number, and quantity).

Maintaining the object orientation, the line editor allows the user to move intuitively along the stations of the line, as if walking along the factory floor. At each station, the user can see a list of the assigned tasks together with their processing times and the relative times at which they are to be started and finished; in addition there are displayed statistics on the work load at the station, including total work assigned and percentage utilization.

The line editor and the product editor are only a keystroke apart, so that the user can select a task in the product editor, and then jump to the line editor to see the station to which the task has been assigned and its place in the schedule of work at that station. Similarly, in the line editor the user can select a task at some station and then jump to the product editor to see detailed information about that task.

From within the line editor the user can rebalance the line at a keystroke. We tried to make rebalancing as fast and unobtrusive as a word processor reformatting text. Balancing a two-sided line with the 148 tasks of the example in the appendix required less than 1 s on a personal computer (in this case, a 386-based machine running under MS-DOS).

Finally, the user can print many types of reports, including a schedule of tasks to be performed and all parts required at each station.

4. The line-balancing algorithm

At the very least, a practical algorithm for balancing assembly lines must accomplish three things.

- Minimize the number of stations for a given cycle time.
- Observe positional and zoning constraints (that is, restrictions on where tasks can be placed),
- Concentrate idle time at one or a few stations, so that waste is exposed and can be attacked.

Most line-balancing algorithms, however, have been designed to address only the first requirement. For example, there are many effective optimum-finding algorithms for line-balancing (Baybars 1986, Johnson 1988, Talbot and Patterson 1984); but we judged them inappropriate to this application for several reasons. Most immediately, the data on task times are insufficiently precise to justify optimization. Naturally, since the workers are humans and not robots, the actual time required by a task varies each time it is performed. Moreover, this variance is exacerbated by the frequent rebalancing of the lines, wherein a worker might have a different set of tasks from week to week. Because of this variance, the manufacturer calculates task times by averaging many observations and then adding 20%.

Another reason for not using optimum-finding algorithms is that they are generally slow. Indeed, on an IBM 370/158-1 mainframe computer the best algorithms required anywhere from 1 s to several minutes to find an optimum balance for only 40 tasks (Baybars 1986). This falls considerably far short of our requirement that the algorithm be fast enough to support intense user interaction on a personal computer.

The recent branch-and-bound algorithm of Johnson (1988) seems to be the fastest of the optimum-finding techniques and might support interactive line-balancing, except that it has other limitations. First, the algorithm achieves its speed by assuming a static set of task data that has been preprocessed. This is useful only where the data does not change much, which has not been the case in any of the lines considered here. Moreover, some of the fathoming rules cannot be applied if certain tasks must be restricted to specified stations, which is likely to be the case in any practical problem. Without these fathoming rules, the algorithm can be expected to perform more slowly. Finally, this algorithm is for one-sided lines and it is not clear that the ideas can be extended to two-sided lines.

Since the standard models of line-balancing fail to capture many of the complexities of the shop floor, it was important to choose an algorithm that supports the quick, incremental building of solutions. The user should be able to fix part of the solution and then have the line-balancing algorithm instantaneously fill in the remainder. However, it is important that the system finish the remainder in a way that can be understood and even anticipated by the user, so that he learns as he works toward an acceptable solution. This means that the algorithm must enable the user to understand and anticipate the effects of the changes he makes. Ideally the algorithm should obey a principle of 'proportional response', so that when the user makes a small change to the problem, he will see a correspondingly small, localized change to the solution. However, optimum solutions, besides being expensive to find, are rigid and fragile: if the user changes some aspect of the problem description, a new solution must be rebuilt from scratch, which can be very time-consuming. Furthermore, the new solution might be very different from the previous one. This means that the user has no consistent sustained control from solution to solution and so cannot build toward a goal.

The standard line-balancing algorithms assign tasks to stations based purely on their processing times. This ignores the fact that some tasks should be assigned to the same station. There are several reasons why tasks might need to be kept together. For example, on one line some tasks required the vehicle to be turned upside-down; it was helpful to keep those tasks together to reduce the vehicle must be inverted. It is also helpful to group tasks that are related in meaning. For example, one series of tasks included loading several batteries into an electric vehicle, clamping the batteries to the frame of the vehicle, cabling the batteries, and then tying the cables together. These tasks can be assigned to different stations if necessary, but it is preferable to have as much of the sequence as possible performed by a single worker. When the tasks at a station are organized by some logic, such as the achieving of a subgoal, the worker can more quickly learn his tasks and can perform them more dependably and faster.

Presumably it is possible to enlarge an optimization model to account for such relations among tasks, but this was discounted on several grounds. First the resulting model would be so unwieldy as to be no longer effectively solvable, and so self-defeating. More importantly, the engineers felt that deciding which tasks were related would take too much time. (Simply gathering data on the precedence requirements and processing times that appear in the appendix required >150 h of an engineer's time, a considerable cost in a lean shop.) Another disincentive to undertake this is that 'relatedness' is not well-defined: For example, should tasks be organized by subgoal? By use of a common tool? Instead, the engineers preferred to formalize as little data as necessary, and then adjust line balances within the program to accommodate additional, unformalized objectives.

We also ruled out some otherwise effective algorithms as insufficiently flexible because they allowed no natural way to explore alternative solutions. This was an important and necessary capability in our project since there are some constraints and goals that are not captured by standard models of assembly line balancing (for example, that some tasks are related, that some workers are more adept at some tasks than others, or that some tasks have preferred stations). Consequently, the user must be able to quickly and easily move among good alternative solutions.

Following the engineering maxim of using the simplest adequate tool, we implemented a 'minimal' heuristic that produces acceptably good solutions with little effort. This heuristic is both fast enough to be interactive and flexible enough to allow the user to easily hunt through alternative solutions. Furthermore, the logic of the heuristic can be extended easily to accommodate real-world complexities not captured by the standard models. The heuristic we implemented is a modified version of 'first fit', which is a familiar heuristic for bin-packing problems (Garey and Johnson 1981, Wee and Magazine 1982). Our version of first fit works by accepting as input a list of the tasks, together with their times, side of line restrictions, and precedence constraints. The iterative step (Table 1 is to identify the next task on the list, all of whose predecessors have already been assigned to stations; and then to schedule it at the first possible station and the earliest possible time.

Our version of first fit is a little more complicated than the familiar version since it must look for opportunities to schedule a task during idle periods in the midst of the schedule as well as at the end of the schedule at a station. For a task that is required to be assigned to, say, the left side of the line, the heuristic determines the

repeat

1. Find the next task in the list, all of whose predecessors have been assigned;
2. Find the last position i to which a predecessor has been assigned, and for that position find the latest finish time t of any predecessor;
3. Beginning at position i and time t find the first place in a schedule into which the task can be inserted, observing the side-of-line restriction of the task and the work limits at the stations;

until all tasks have been assigned;

Table 1. A first-fit algorithm to assign tasks to stations in a two-sided line.

last station to which an immediate predecessor is assigned, and the time at which the last predecessor at that station completes processing. Then the heuristic searches through the remainder of the cycle time at the left station for an available block of time in which to schedule the task. If none is found, the heuristic continues the search beginning at time 0 of the next station on the left, until the task can be scheduled. Similarly, when scheduling a task that must be assigned to both sides of the line simultaneously, the heuristic performs the same sort of search, but in parallel, looking for a sufficiently large block of time during which two opposite station are both idle. For tasks that can be assigned to either side of the line, the heuristic again operates similarly, except that it searches the schedule of the opposite station before searching that of the next station.

Several of the heuristics discussed by Talbot *et al.* (1986) assign tasks according to a similar logic to stations on a one-sided line; but whereas the sequence in which the tasks appear in our input list is arbitrary, these others work from special lists that are expected to produce generally better line balances. Such lists include those in which the tasks are sorted in advance according to, for example, largest task times, largest number of immediate successor tasks, largest total number of successor tasks, largest ranked positional weight, or other, more complicated measures. While such heuristics can be expected to produce slightly more effective line balances, they suffer from some of the weaknesses of optimum-finding methods: they offer no simple and natural way to search over alternative line balances, and they fail to assign related tasks together.

In contrast, we designed our program to assign tasks from an arbitrary list, and then made it easy for the user to change the list, and so produce alternative solutions. Furthermore, changing the list results in generally predictable changes in the line balance, so that, for example, moving a task toward the front of the list results in its being assigned to an earlier station. Even though the list is arbitrary, we expect good solutions; those features that seem necessary for worst-case behaviour (such as pathological precedence constraints) do not hold. (See the appendix for a more complete discussion of data typical to this project.)

The line-balancing program handles positional and zoning constraints indirectly. The list from which the tasks are scheduled is initially the order in which the tasks have been input. This is partly for convenience, and partly in the expectation that the user will naturally tend to group related tasks when entering them in the database. When tasks appear together in the list, they tend to be assigned to the same station. The user can easily change the position of a task in the list, and so change the station to which it is assigned. This enables him to direct special tasks to

predetermined stations, and related groups of tasks to the same station. For example, to move a selected task to a position later in the line, the user presses a key to indicate 'later'. Then that task and all of its descendants in the precedence network are moved down the list so that in the subsequent rebalance the task is assigned to the next later position.

Changing the line-balance by changing the position of tasks in the list is a useful but weak control over the behaviour of the program. The program also provides more direct control by allowing the user to 'lock' a task to a particular station and time. Subsequent rebalancing respects any locked assignment. The user can lock an assignment in either of two ways. As the user views a task within the product editor, he can press a key that will open a dialog box in which to specify a desired assignment. If that assignment is consistent with all other locked assignments, then it is accepted; otherwise the user is warned, and the conflict identified. As the user views a station within the line editor, he can select any task at that station and lock or unlock its current assignment at the press of a key.

In the lines that we balanced there were 2–20 tasks that required assignment to specific stations. We were generally able to reduce this to no more than 10 tasks by aggregating tasks required to be at the same station, after which it was fairly simple to adjust the list of tasks to achieve the required placements.

Line-balancing algorithms of the first-fit type, such as ours, usually load the first stations more heavily than the last stations and so tends to concentrate any idle time at the last stations. This is important because it makes inefficiencies evident and attackable since they are localized. (In contrast, it would be much more difficult to eliminate waste that was spread among workstations.) Note that other algorithms, such as branch-and-bound, cannot be relied upon to concentrate idle time.

Our heuristic also allows fine-tuning of the balance through the following mechanism: The program allows the user to set an independent limit on the work allowed at each individual station. By reducing the work at a station the user can selectively 'squeeze' work from a station to subsequent stations by lowering the work limit at that station. The limit for a station can even be set to 0 so that it will not be assigned any tasks. This is necessary when, for example, there are obstructions, like supporting pillars, that prevent some stations from having a twin on the opposite side. Alternatively, the work limit can be set to a value greater than the cycle time at stations to which management will assign more than one worker.

5. Quality of the balances

For the data of the actual lines, the heuristic reliably produced balances that required no more than 1–2 stations beyond the very weak lower-bound on optimum derived by dividing the total work content of the product by the intended cycle time of the line. Note that this lower bound ignores all precedence constraints, assumes that all tasks can be assigned to either side of the line, and assumes that all tasks are continuously partitionable. This is consistent with Talbot *et al.* (1986), which reported that balancing a one-sided line with first fit from an arbitrary list required 8–10% more stations than optimum.

The most dramatic success of the program was to reduce the staffing of a line with a fixed cycle time from 59 workers to 53. This was almost certainly optimum since a perfect, continuous partition of the work would have required 52 workers.

An important strength of our program is its convenient interaction, so that if the heuristic does not produce a satisfactorily balanced line, the user can easily

intervene. We reliably produced optimal lines within a few minutes by adjusting the sequence in which tasks are assigned to stations and then rebalancing the line. For example, for the product in the appendix and with a cycle time of 4·00 min, the heuristic gave an initial balance that required 18 stations, as given in Fig. 5. In examining the line with the line editor, we quickly saw how to improve the balance by changing the sequence in which tasks were assigned to stations. Moving three tasks forward in the sequence gave a balance that reduced the stations to 16, as shown in Fig. 6. Finally, we consolidated the tasks at the end of the line by setting the work limit at station 8-R to 0, which caused its work to be reassigned to station 8-L. This gave a balance that required only 15 stations, as shown in Fig. 7.

The balance of Fig. 7 is optimal, and required only a few minutes to find. Moreover, this balance concentrates idle time at a single station. Management took advantage of this to assign the last worker some additional tasks, such as restocking parts and performing quality checks and rework. They also recognized a clear incentive to improve the manufacturability of the vehicle so that the last worker could be eventually eliminated.

6. Conclusions

When measured against the ideal there are some shortcomings in our line balancing program. In particular, the users want a still more intuitive way to directly assign selected tasks to stations. They would like to be able to require an assignment by simply pointing at a task and at a station. We can do this for a one-sided line since the computations are much easier; but this feature is impractically slow for two-sided lines because of the additional complexity. Our approximation of this feature requires the user to specify both the station and the time for the task. Unfortunately the appropriate time might not be obvious. It would be nice if the program could determine it, but that does not seem practical.

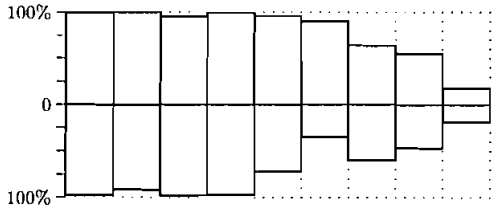

Figure 5. Utilizations of the 18 stations in the initial balance.

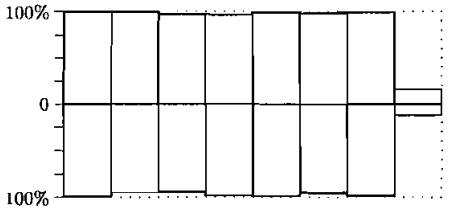

Figure 6. Utilizations of the 16 stations required after changing the sequence of tasks and rebalancing.

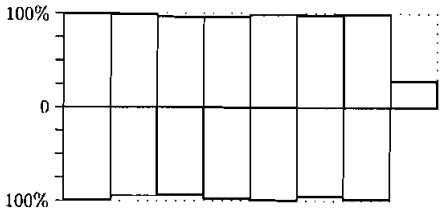

Figure 7. Utilizations of the 15 stations required after setting the work limit of station 8-right to 0. The balance is optimal and, moreover, concentrates idle time at one position.

Another difficulty is one that is inherent in combinatorial problems: the lack of proportional response. Our algorithm does fairly well in this regard since if the user changes the work standards for a task and then rebalances the line, the assignments of only the descendant tasks are affected. Thus part of the previous solution remains intact. It is not clear that any reasonable heuristic could provide more proportional response for this problem. Nevertheless, the users wished for it.

An immediate benefit of the program is that rebalancing is now fast and convenient. Before out program an engineer had balanced each line by entering the data into a spreadsheet and then assigning tasks to stations by moving rows of data. It took about 3 months of intermittent effort to satisfactorily balance one line. Furthermore this was prone to error because of the difficulty in keeping track of precedence constraints.

A more profound benefit has been to enable the users to manage work standards. In particular, users can now evaluate the effect on labour requirements of proposed changes such as changes in the production rate or design changes that might alter task times or precedence.

Finally, since the line-balancing program also maintains a database of parts, it generates a report of which parts are required at each station and rate at which they are required. This has made it easier to plan an effective strategy for restocking the work stations.

Acknowledgments

The author thanks S. T. Hackman, H. D. Ratliff, J. H. Vande Vate, and C. A. Yano for helpful comments. This work has been partially supported by the National Science Foundation (DDM-9101581), and by the Office of Naval Research (N00014-89-J-1571).

Appendix

In emphasizing the need for new test problems, Baybars (1986) observed that 'after a quarter of a century, the 70-task problem of Tonge (1961) remains as the benchmark test problem'. We contribute the following 148-task problem on which researchers can hone their algorithms. As far as we know, these are the first data for a real assembly line to appear in the literature in over 30 years, and are the first published data on a two-sided line. They conform remarkably little to typical assumptions for generating random test problems.

These are the tasks required to produce a small vehicle. There are 148 tasks, totalling 56·34 min of work. Unlike typical random test problems, the task times are not at all uniformly distributed, as shown in Fig. A.1.

This distribution helps explain why it is not difficult to find good line balances: the few long tasks require that the cycle time be rather large relative to most task times, so the problem approximates a continuous one.

Of the 148 tasks, 87 can be assigned to either side of the line; 35 require assignment to the left side of the line, and 26 require assignment to the right. The left and right sides have different numbers of tasks because of asymmetries in the product, such as the steering wheel on the left. Another asymmetry is that for some tasks the processing time is close but not equal to that of its mirror image task on the other side of the vehicle (tasks 21 and 22 for example). This is because a right-handed worker might find one of the tasks more awkward, and so take longer.

The 'order strength' of a network is the total number of precedences in the transitive closure of the network divided by the largest possible number of precedences ($n(n+1)/2$ for n tasks). Among the tasks of this product there are 175 non-redundant precedence constraints and the order strength of the precedence network is 0·258. This is rather low by comparison to the randomly generated networks on which others have tested algorithms. This is attributable in part to the relatively large size of the product; since the tasks are relatively localized, they do not interfere much with each other. (By comparison, the precedence networks for the differential and for the drive axle, which were built on smaller separate lines, each had order strength approximately 0·5.)

The precedence network reflects some of the symmetry of the product, as can be seen in Fig. A.2. This sort of structure is unlikely to be generated by the standard ways of producing random test problems.

A final complication for this line is that some tasks must be assigned to certain absolute physical locations because specialized equipment is permanently mounted there. These restrictions are given in Fig. A.3 and can be interpreted as follows. The area available for the assembly line is long enough to be occupied simultaneously by 34 vehicles, so we can imagine 34 positions along the line at which there might be pairs of stations. We refer to the absolute physical locations at the ith position, left and right, along the line as locations i-L and i-R. At each location there might or

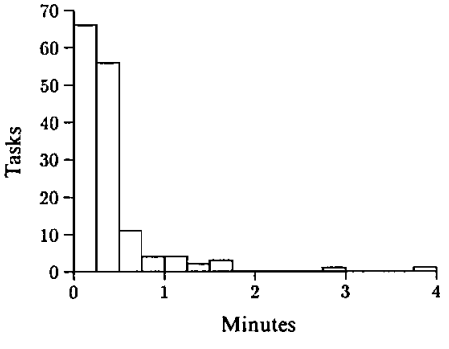

Figure A.1. Distribution of task times. The average time is 0·38 min, but the longest task requires 3·83 min.

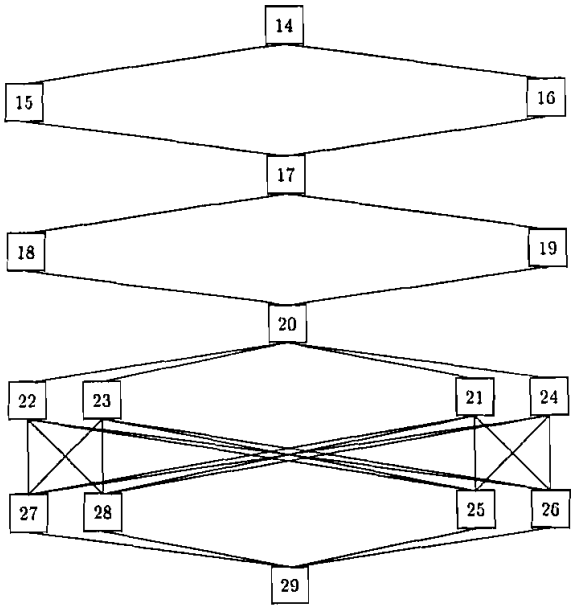

Figure A.2. A fragment from the precedence network. Its symmetry reflects that of the product, a utility vehicle. (Tasks on the left must be assigned to the left side of the line; tasks on the right must be assigned to the right; and tasks in the middle can be assigned to either side. All arcs are directed downward.)

Tasks	Station	Specialty
43, 44, 45, 46, 47	7-L	mate differential
35, 37, 38, 39, 40	7-R	mate differential
76, 77, 78, 79, 80, 81	13-L or -R	brake work
99, 100	17-R	rubber work
106, 107, 108, 109	21-L or -R	brake work
112, 113	24-L	install battery
141, 144	30-L	rubber work
145, 142	30-R	rubber work

Figure A.3. Required assignments of tasks to stations.

might not be a station, depending on how the line is physically laid out. When certain tasks are required to be at designated physical locations, then the balanced line must satisfy the following:

- all tasks required to be at the same physical location are assigned to the same station;
- all tasks required to be at opposite physical locations (i-R and i-L) are assigned to opposite stations;
- all tasks required to be at later physical locations are assigned to later stations;
- no more than $j-i+1$ physical positions strictly between locations i and j (for $i<j$) have stations with tasks assigned.

For example, the tasks required to be assigned to physical location 7-L need not be assigned to the seventh station in the line. However, the station to which they are assigned must be opposite the station to which are assigned those tasks required to be at location 7-R and before the station to which are assigned those tasks required to be at location 13. In addition, there can be no more than five positions between the tasks required to be at location 7 and those required to be at location 13 since there is room for no more than five vehicles strictly between positions 7 and 13.

Finally, the precedence and estimated task times are given in the list below.

Each task is identified by a number. 'Side' tells whether the task must be assigned to a special side of the line: 'L' = left side only; 'R' = right side only; 'E' = either side; 'B' = both sides simultaneously. Time is given in decimal minutes.

Task	side	time	precedes
1	E	0·16	5,6,7,8
2	E	0·30	3
3	E	0·07	4,5,6,7
4	E	0·47	8
5	E	0·29	14
6	E	0·08	9
7	E	0·39	14
8	E	0·37	10
9	E	0·32	14
10	E	0·29	14
11	E	0·17	12
12	E	0·11	13
13	E	0·32	—
14	E	0·15	15,16
15	L	0·53	17
16	R	0·53	17
17	E	0·08	18,19
18	L	0·24	20
19	R	0·24	20
20	E	0·08	21,22,23,24
21	R	0·07	25,26,27,28
22	L	0·08	25,26,27,28
23	L	0·14	25,26,27,28
24	R	0·13	25,26,27,28
25	R	0·10	29
26	R	0·25	29
27	L	0·11	29
28	L	0·25	29
29	E	0·11	31
30	R	0·29	—
31	E	0·25	36
32	L	0·10	34
33	R	0·14	35
34	L	0·41	36
35	R	0·42	36
36	R	0·47	37
37	R	0·07	38,45
38	R	0·80	39
39	R	0·07	40
40	R	0·41	41,48,55
41	R	0·47	—
42	L	0·16	43
43	L	0·32	44
44	L	0·66	—
45	L	0·80	46
46	L	0·07	47
47	L	0·41	48,49,55
48	E	0·13	—
49	L	0·47	—
50	E	0·33	51
51	L	0·34	53,69
52	L	0·11	53
53	L	1·18	—
54	L	0·25	133
55	R	0·07	54,72,76,87,88
56	E	0·28	73
57	L	0·12	79
58	L	0·52	84,86
59	E	0·14	75,87
60	E	0·03	—
61	E	0·03	62
62	E	0·08	63
63	E	0·16	67
64	R	0·33	65,71,72
65	E	0·08	66,99
66	E	0·18	67
67	E	0·10	68
68	E	0·14	95,98
69	R	0·28	82
70	R	0·11	71
71	R	1·18	—
72	R	0·25	134
73	E	0·40	84,86,87,88,96
74	E	0·40	75
75	E	1·01	88,97
76	E	0·05	77
77	E	0·28	78
78	E	0·08	79
79	E	2·81	80
80	E	0·07	81
81	E	0·26	106
82	E	0·10	83,89,143,146
83	E	0·21	—
84	E	0·26	85
85	E	0·20	—
86	E	0·21	—
87	E	0·47	—
88	E	0·23	111

89	E	0·13	90	119	E	0·55	—
90	E	0·19	79	120	E	0·31	121
91	E	1·15	105	121	E	0·32	122
92	E	0·35	135	122	E	0·26	126
93	L	0·26	—	123	E	0·19	124
94	E	0·46	—	124	E	0·14	125
95	E	0·20	101	125	E	0·19	—
96	E	0·31	104	126	E	0·48	—
97	E	0·19	—	127	E	0·55	—
98	E	0·34	101	128	L	0·08	129
99	E	0·51	100	129	L	0·11	130
100	E	0·39	101	130	L	0·27	131,137
101	E	0·30	102,103	131	L	0·18	—
102	E	0·26	127	132	E	0·36	135
103	E	0·13	127	133	L	0·23	135
104	E	0·45	—	134	R	0·20	135
105	E	0·58	119	135	E	0·46	136
106	E	0·28	107	136	E	0·64	—
107	E	0·08	108	137	L	0·22	—
108	E	3·83	109	138	E	0·15	139
109	E	0·40	110	139	E	0·34	140
110	E	0·34	—	140	E	0·22	—
111	E	0·23	112	141	L	1·51	142
112	L	1·62	113	142	R	1·48	143,146,147,148
113	L	0·11	114,116,120,123,128	143	L	0·64	—
114	E	0·19	115	144	L	1·70	145
115	E	0·14	125	145	R	1·37	147,148
116	E	0·31	117	146	R	0·64	—
117	E	0·32	118	147	L	0·78	—
118	E	0·26	126	148	R	0·78	

References

BAYBARS, I., 1986, Survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, **32**(8), 909–932.

GAREY, M. B., and JOHNSON, D. S., 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (San Francisco: W. H. Freeman).

GAREY, M. B., and JOHNSON, D. S., 1981, Approximate algorithms for the bin-packing problem: a survey, in *Analysis and Design of Algorithms in Combinatorial Optimization*, G. Auselio and M. Lucertini (eds), (New York: Springer), pp. 178–190.

JOHNSON, R. V., 1988, Optimally balancing large assembly lines with 'FABLE'. *Management Science*, **34**(2), 240–253.

PINTO, P. A., DANNENBRING, D. G., and KHUMAWALA, B. M., 1981, Branch and bound and heuristic procedures for assembly line balancing with parallel stations. *International Journal of Production Research*, **19**, 565–576.

TALBOT, B. F., and PATTERSON, J. H., 1984, An integer programming algorithm with network cuts solving the assembly line balancing problem. *Management Science*, **30**(1), 85–99.

TALBOT, B. F., PATTERSON, J. H., and GEHRLEIN, W. V., 1986, A comparative evaluation of heuristic line balancing techniques. *Management Science*, **32**(4), 430–454.

WEE, T. S., and MAGAZINE, M. J., 1982, Assembly line balancing as generalized bin packing. *Operations Research Letters*, **1**, 56–58.