AN INTERACTIVE PROGRAM TO BALANCE ASSEMBLY LINES

John J. Bartholdi, III *

1992; April 3, 2003

Abstract

We describe the development of a program to balance 1- or 2-sided assembly lines for a manufacturer of utility vehicles. The program is highly interactive, runs on a personal computer, and is in use now. We also discuss some theoretical properties of 2-sided lines. Finally, we include the data for a real assembly line.

Key words: assembly lines, line balancing, heuristics, software

*Supported in part by the National Science Foundation (DDM-9101581), and by the Office of Naval Research (N00014-89-J-1571). Address: School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332

1 Introduction

A manufacturer of small utility vehicles rebalances its several very flexible 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 labor. 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. This flexibility is important to the manufacturer: their main product is a golf cart, so that demand is highly seasonal.

Each line has 2 sides, left and right, with stations in pairs directly opposite each other 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 testing the effects of design changes on the manufacturing process. For example, an engineer might redesign a fender, thereby changing some of the tasks associated with installing it; then using the program to rebalance the line with the hypothetical changes, he could see how the line might change.

There is a rich literature on assembly line balancing. However, most of it concerns algorithms to balance an idealized 1-sided line. Our paper complements this literature by describing the inevitable compromises made in translating abstract models to a commercial product in use on the shop floor. In addition

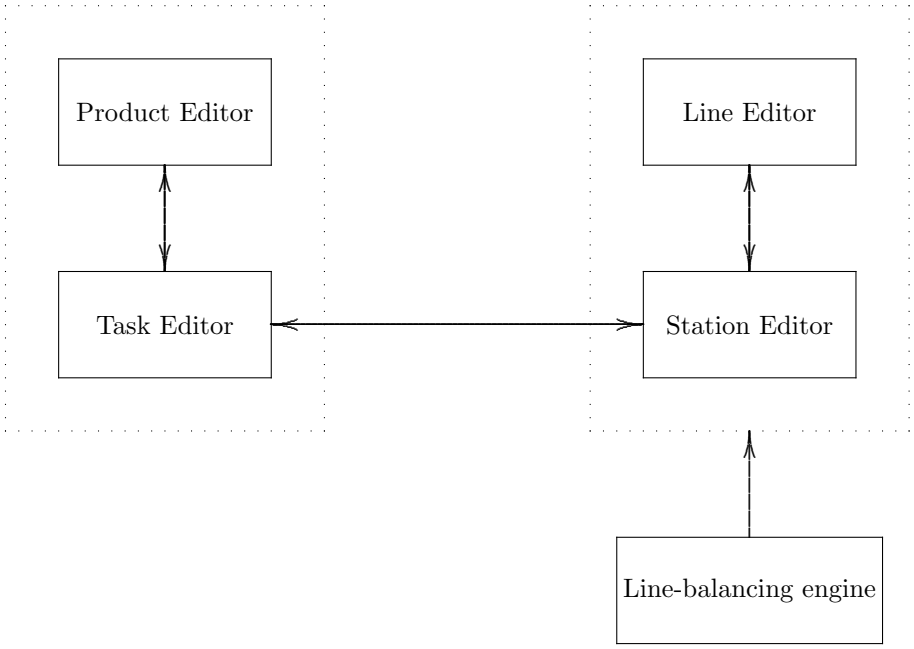

Figure 1: General structure of the program.

we raise some new issues in balancing a 2-sided line. Finally, we include the data for a real assembly line, which to our knowledge is the first to appear in the open literature in almost 20 years.

2 General structure of the program

The program consists of two main functional units, the Product/Task Editor and the Line/Station Editor. The Product Editor manages the data of a product, and the Line Editor manages the data of an assembly line. Both editors were designed to build and adjust solutions incrementally. The user can toggle back and forth between the two Editors, as suggested by Figure 1.

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 work stations (rather than only with the text decribing them).

For example, any such object can be selected, displayed, printed, modified, or deleted.

The Product Editor provides access to the tasks according to three different metaphors. In the first metaphor, each task is associated with an "index card" which contains its description; all the index cards are arranged in a stack through which the user moves by simple keystrokes. Alternatively, the user can imagine each task to be the text which constitutes its description, and can locate tasks based on that text. For example, he can pop open a window and ask the program to search for the next task whose description includes the word "fender". In the third metaphor 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. These three ways of viewing the tasks are seamless; the user can adopt any one of them at any time. Also, at any time the user can edit any of the data associated with a task, which includes the following.

- a detailed text description of the task (which can be arbitrarily long);
- an abbreviated text description or "name" (used for certain screen displays and printed reports in which space is limited);
- 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 percent 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 task in the work schedule of a station on the line. 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 2-sided line with the 148 tasks of the Appendix requires less than 1 second on a personal computer (in this case, a 386-based machine running under MS-DOS).

When the user has balanced the line to his satisfaction, he can print many types of reports, including a schedule of tasks to be performed and all parts required at each station. The general rule is that the user can print a report about whatever he is currently viewing, whether it be the entire product or a single task, the entire assembly line or a single station.

3 Balancing a 2-sided line

While our program will balance both 1- and 2-sided lines, certain features of 2-sided lines are especially interesting (and apparently unremarked on heretofore). On a 2-sided line the workers at each pair of opposite stations work in parallel on one individual product. This is different from working in parallel on individual tasks. It is also different from working in parallel on multiple products, as in [5], wherein stations are duplicated, with each assigned the same set of tasks. It is practical for two people work in parallel on an individual product only when they will not interfere with each other; thus it is more likely to be practical for large products like vehicles than for small ones like electric drills.

A 2-sided line offers several advantages over a 1-sided line. On a 2-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 2-sided line can be more space-efficient since the line can be shorter in length than a 1-sided line. (In fact, as we show shortly, a 2-sided line can require fewer stations than a 1-sided line.) A shorter line can reduce material handling costs since it mitigates the need for workers to manœuver tools, parts, or the product. 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 2-sided line in fact delivers advantages over a 1-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 "string" in which task A must immediately precede task B, which must precede task C, and so on. For such a set of tasks, any 2-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 2-sided line offers no advantages over a 1-sided line in this example.

On the other hand, a 2-sided line can require fewer stations than a 1-sided line, as shown by the example in Figure 2. For a cycle time of 10, a 2-sided line requires 2 stations (one with tasks A, C, and the other with tasks B, D), while a 1-sided line requires 3 stations (one with tasks A, B, one with task C, and one with task D). This illustrates the sort of precedence for which 2-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 its mirror image must precede the mirror image of task j.

The preceding examples show that 1-sided and 2-sided lines can require different numbers of work stations to produce the same product at the same rate. The following theorem shows that a 2-sided line can require as few as two-thirds as many stations as a 1-sided line. If all stations are the same size, then this means that a 2-sided line can be as little as one-third the length of

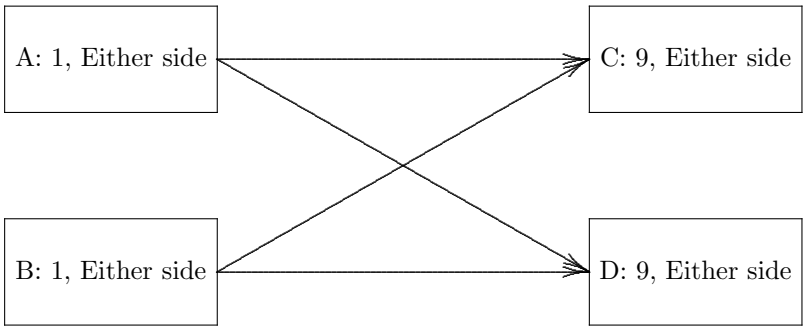

Figure 2: For this set of tasks and a cycle time of 10, a 2-sided line requires fewer stations than a 1-sided line.

a 1-sided line for the same product. Alternatively, with the same number of stations, a 2-sided line can achieve a production rate up to 3/2 times that of a 1-sided line.

Theorem 1. *Consider a set of tasks for which any task can be assigned to either side of the line. For any fixed cycle time let z_2 be the minimum number of stations required in a 2-sided assembly line, and let z_1 be the minimum number required in a 1-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 1-sided line can be viewed as a 2-sided line with some stations unused.

To show that $(2/3)z_1 \leq z_2$, consider any 2-sided line, and within it, any opposite pair of stations. It is possible to reassign the tasks at these 2 stations to no more than 3 consecutive stations devoted solely to these tasks on a 1-sided line; for if 4 or more stations were required to contain these tasks, then since the total work content at each consecutive pair of stations on the 1-sided line must exceed the cycle time, there must have been more than 2 stations worth of work. This contradicts the assumption that these tasks were feasibly assigned to a pair of opposite stations on a 2-sided line. Since the tasks at each pair of stations on any 2-sided line can be reassigned to no more than three stations on a 1-sided line, $z_1 \leq (3/2)z_2$.

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

theorem. □

(Note that for a real product a 2-sided line can actually require *fewer* than two-thirds the number of stations of a 1-sided line, since, as mentioned earlier, some tasks can in effect be shorter on a 2-sided line since they can require less setup time.)

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 1-sided line.

To balance a 2-sided line is a difficult problem. Of course it is formally difficult in the sense of being $\mathcal{NP}$-complete [2] since the same is true of balancing a 1-sided line. It seems even more challenging to produce good balances for 2-sided lines since there is an additional level of complexity. As for a 1-sided line, there is the difficult problem of assigning tasks to positions along the line (that is, stations on a 1-sided line, or pairs of stations on a 2-sided line). But for a 2-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 Figure 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 2-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 2-sided line, a station can be intermittently busy and idle.

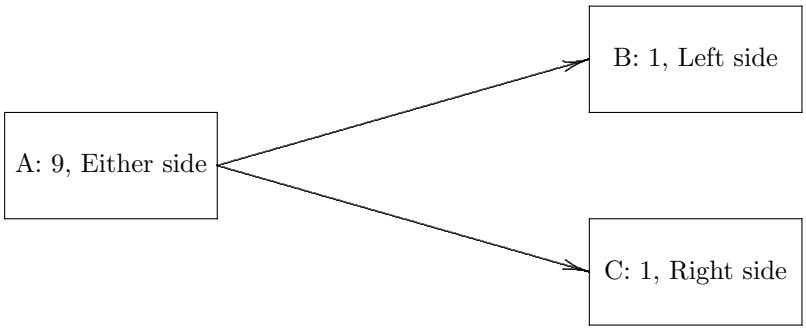

Figure 3: When this set of tasks is assigned to a 2-sided line with a cycle time of 10, one station will have forced idle time in the midst of its schedule.

(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.)

In designing our program we ignored the possibility of interference between tasks and relied on the expectation that in the final balance there would not be "too much" idle time in the midst of the schedule of any station. This expectation seemed justified because the precedence constraints were rather sparse, so there was some flexibility in scheduling tasks to fill any such idle time. In addition, about 10% of the tasks had no immediate predecessors and might therefore be scheduled to fill any such idle time. This approach worked well. Idle time was required in the midst of the schedule for a station only very occasionally (perhaps once in a line balance), and then it was always quite small, on the order of the shortest task times. This was small enough to be absorbed by the variance of the task times.

4 The line-balancing algorithm

4.1 Design issues

There are many effective optimum-finding algorithms for line-balancing [1, 4, 6]; however, we judged them inappropriate to this application. We required

an algorithm that is fast enough to support intense user interaction, and we expected optimum-finding algorithms be too time-consuming to be sufficiently interactive on the current generation of personal computers. Indeed, on an IBM 370/158-1 mainframe computer the best algorithms required anywhere from 1 second to several minutes to find an optimum balance for only 40 tasks [1].

Another reason for not using optimum-finding algorithms is that 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 issue by which algorithms were evaluated was whether they supported the incremental building of solutions. The user should be able to fix part of the solution and then have the system 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 heuristic must enable the user to understand and anticipate the effects of the changes he makes. Ideally the algorithm should obey the principle of "proportional reponse", 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 are rigid, fragile, and expensive to find and maintain. If the user changes some constraint, a new solution must be rebuilt from scratch, which can be very time-consuming. Furthermore, the new solution might be very different. This means that the user has no consistent, sustained control from solution to solution and so cannot build toward a goal.

Many algorithms, both optimum-finding and heuristic, were judged inappropriate because they are based on a model that is too narrow for the circumstances of this problem. For example they assign tasks to stations based purely on their processing times. This ignores the fact that some tasks are related in

meaning, and ought to be grouped together when possible. 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 sequence of tasks. (This was important in our project since the lines are rebalanced so frequently.) In addition, the worker can perform his tasks more dependably and more quickly.

Of course 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 over 150 hours 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.

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 2-sided line

4.2 An appropriate heuristic

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 [3, 8]. 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 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 stations 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 [7] assign tasks according to a similar logic to stations on a 1-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 behavior (such as pathological precedence constraints) do not hold. (See the Appendix for a more complete discussion of data typical to this project.)

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 behavior 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 by pressing 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 achieve the desired placements.

Line-balancing algorithms of the "First Fit" type usually produce a line in which the first stations are loaded more heavily than the last stations. Our heuristic shares this defect but it can be ameliorated by iteratively re-balancing the line: first balance the line at the desired cycle time and note the number of stations required; then repeatedly reduce the cycle time and rebalance the line, until the cycle time is as small as can be without increasing the required number of stations beyond that of the initial solution. This process "squeezes" tasks from heavily loaded stations to lightly loaded stations. Since the line can

be rebalanced quickly, this entire process takes less than a minute. The final balance gives the assignment of tasks to stations; it is more level than the initial solution, uses no more stations, and meets the desired cycle time.

(It actually proved useful to this application to have balances that gave slightly less work to the later stations. Since the line was slightly asynchronous, the later stations experienced more variance in the arrival times of vehicles. Smaller work load helped protect the later stations against surges in the line.)

To give the user a more control over the levelling we added an additional capability: the user can set an independent limit for each individual station on the amount of work assigned there. This allows the user to selectively "squeeze" tasks 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.

5 Quality of the balances

In the worst case our heuristic can produce a line with up to twice the minimum required number of stations, as seems to be characteristic of heuristics based on the paradigm of bin-packing from a list [8]. However, as is typically the case, in practice the heuristic worked much better. 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 time by the intended cycle time. Note that this lower bound in effect ignores all precedence constraints, assumes that all tasks can be assigned to either side of the line, and assumes that all tasks are infinitely partitionable. This is consistent with [7], which reported that balancing a 1-sided line with First Fit from an arbitrary list required 8–10% more stations than optimum.

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

Left	100	100	95	99	96	90	64	55	18
Position	1	2	3	4	5	6	7	8	9
Right	98	92	98	97	72	35	60	47	18

Figure 4: Utilizations of the stations in the initial balance. All utilizations are given in percentage of cycle time.

Left	100	99	97	97	99	98	99	16
Position	1	2	3	4	5	6	7	8
Right	99	95	94	98	99	95	98	12

Figure 5: Utilizations after changing the sequence of tasks and rebalancing.

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 minutes, the heuristic gave an initial balance that required 18 stations, as given in Figure 4. 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 Figure 5. 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 Figure 6.

The balance of Figure 6 is optimal, and required only a few minutes to find. Moreover, the sequence of tasks that produced this optimal solution also produced—without further adjustment—optimal or very-nearly optimal balances for other cycle times as well.

In the balance of Figure 6, the last station is lightly utilized. If desired,

Left	100	99	97	97	99	98	99	28
Position	1	2	3	4	5	6	7	8
Right	99	95	94	98	99	95	98	

Figure 6: Utilizations after setting the work limit of station 8-Right to 0.

one could better balance the utilization among stations by lowering the work limits at earlier stations and rebalancing the line. Instead, the manufacturer preferred to assign the last worker some additional tasks such as performing quality checks and doing whatever rework might be necessary.

While the line balances were optimal or nearly so, in some ways this was not the most important contribution of the program. As important is the savings in time to balance a line. Prior to our 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.

6 Additional features of the program

The program is written in Modula-2, and so is, as the name suggests, highly modular [9]. For example, we can easily pull out the current line balancing module, and "plug in" another. This makes the program a useful environment for exploring alternative line-balancing algorithms. It also makes the program a flexible platform that can grow into a more comprehensive system of production and inventory control.

As to be expected, much of the program overhead goes to support the user interface, which implements currently standard programming paradigms, such as pull-down menus, context-sensitive help, windows, and so on. (The line balancing algorithm accounts for only about 800 lines of code out of 20,000.) In addition, much effort goes to manage the data describing a product. Indeed, balancing the line might be viewed as just another way among many in which to query the product data.

It is worth observing that the data for an assembly line are precious. They take considerable effort to gather. If they are lost or become corrupted, the newly-balanced line might fail to meet intended specifications, with potentially expensive consequences. Accordingly, most of our program is devoted to manag-

ing and protecting this data. For example, the program prevents the user from inserting contradictory precedence constraints among tasks. Also, the program asks the user to confirm his intentions before allowing him to delete or change any critical information, such as task times or precedence constraints.

7 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 of assigning 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 1-sided line since the computations are much easier; but this feature is impractically slow for 2-sided lines because of the additional complexity. Our approximation of this feature requires the user to specify a station and a time for the task. This is less convenient because the appropriate time might not be obvious. The program should determine it.

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 data about a task and then rebalances the line, the assignments of all tasks that are appear earlier in the list (and the assignments of all ancestor tasks) remain unaffected. Thus part of the previous solution remains intact. However, later tasks on the list might be assigned very differently. It is not clear that any reasonable heuristic could provide more proportional response for this problem. Nevertheless, the users wished for it.

The most immediate benefit of the program is that rebalancing is now fast and convenient. In addition there have been other unexpected changes. The very fact of having organized so much data and made them manageable is in some ways changing the meaning and use of the data. For example, the data on task times are becoming in effect work standards for the manufacturer. Similarly, the descriptions of tasks and their precedences are becoming engineering standards that guide the design of vehicles. It is now much easier to estimate

the effect of design changes on the production process; it is also easier to ask directly what design changes would help improve production. Finally, having the list of parts needed at each station makes restocking more convenient and more dependable.

Acknowledgements

The author thanks S. T. Hackman, L. F. McGinnis, H. D. Ratliff, J. H. Vande Vate, and C. A. Yano for helpful comments.

References

[1] I. Baybars (1986). "Survey of exact algorithms for the simple assembly line balancing problem", *Management Science* 32(8): 909–932.

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

[3] M. B. Garey and D. S. Johnson (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.), Springer, New York, 178–190.

[4] R. V. Johnson (1988). "Optimally balancing large assembly lines with 'FABLE"', *Management Science* 34(2): 240–253.

[5] P. A. Pinto, D. G. Dannenbring, and B. M. Khumawala (1981). "Branch and bound and heuristic procedures for assembly line balancing with parallel stations", *International Journal of Production Research* 19: 565–576.

[6] B. F. Talbot and J. H. Patterson (1984). "An integer programming algorithm with network cuts solving the assembly line balancing problem", *Management Science* 30(1): 85–99.

[7] B. F. Talbot, J. H. Patterson, and W. V. Gehrlein (1986). "A comparative evaluation of heuristic line balancing techniques", *Management Science* 32(4): 430–454.

[8] T. S. Wee and M. J. Magazine (1982). "Assembly line balancing as generalized bin packing", *Operations Research Letters* 1: 56–58.

[9] N. Wirth (1985). *Programming in Modula-2 (third, corrected edition)*, Springer-Verlag, New York.

Appendix

In emphasizing the need for new test problems, Baybars [1] 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 almost 20 years, and are the first published data on a 2-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 minutes of work. Unlike typical random test problems, the task times are not at all uniformly distributed, as shown in Figure 7.

This distribution helps explain why it is not difficult to find optimal 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 different numbers of tasks for the left and right sides is due to 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,

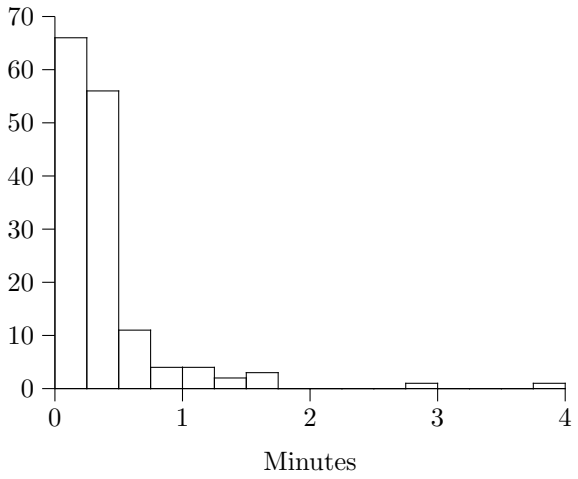

Figure 7: Distribution of task times. The average time is 0.38 minutes, but the longest task requires 3.83 minutes.

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 Figure 8. 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 Figure 9 and can be interpreted as follows. The area available for the assembly line is long enough to be occupied

Figure 8: 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
36, 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 9: Required assignments of tasks to stations.

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 along the line as locations i-L and i-R. At each location there might or 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 5 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 5 vehicles strictly between positions 7 and 13.

Finally, the precedence and estimated task times are given in the list beginning on the following page.

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
90	E	0.19	79
91	E	1.15	105
92	E	0.35	135
93	L	0.26	-
94	E	0.46	-
95	E	0.20	101
96	E	0.31	104
97	E	0.19	-
98	E	0.34	101
99	E	0.51	100
100	E	0.39	101
101	E	0.30	102,103
102	E	0.26	127
103	E	0.13	127
104	E	0.45	-
105	E	0.58	119
106	E	0.28	107
107	E	0.08	108
108	E	3.83	109
109	E	0.40	110
110	E	0.34	-
111	E	0.23	112
112	L	1.62	113
113	L	0.11	114,116,120,123,128
114	E	0.19	115
115	E	0.14	125
116	E	0.31	117
117	E	0.32	118
118	E	0.26	126
119	E	0.55	-
120	E	0.31	121
121	E	0.32	122
122	E	0.26	126
123	E	0.19	124
124	E	0.14	125
125	E	0.19	-
126	E	0.48	-
127	E	0.55	-
128	L	0.08	129
129	L	0.11	130
130	L	0.27	131,137
131	L	0.18	-
132	E	0.36	135
133	L	0.23	135
134	R	0.20	135
135	E	0.46	136
136	E	0.64	-
137	L	0.22	-
138	E	0.15	139
139	E	0.34	140
140	E	0.22	-
141	L	1.51	142
142	R	1.48	143,146,147,148
143	L	0.64	-
144	L	1.70	145
145	R	1.37	147,148
146	R	0.64	-
147	L	0.78	-
148	R	0.78	-