6

Disassembly Line Balancing

Seamus M. McGovern and Surendra M. Gupta

CONTENTS

6.1 Introduction

Manufacturers are increasingly recycling and remanufacturing their post-consumed products due to new, more rigid environmental legislation, increased public awareness, and extended manufacturer responsibility. In addition, the economic attractiveness of reusing products, subassemblies or parts instead of disposing off them has further fueled this effort. *Recycling*

is a process performed to retrieve the material content of used and nonfunctioning products. *Remanufacturing*, however, is an industrial process in which worn-out products are restored to like-new conditions. Thus, remanufacturing provides the quality standards of new products with used parts.

Product recovery seeks to obtain materials and parts from old or outdated products through recycling and remanufacturing to minimize the amount of waste sent to landfills. This includes the reuse of parts and products. There are many attributes of a product that enhance product recovery; examples include: ease of disassembly, modularity, type and compatibility of materials used, material identification markings, and efficient cross-industrial reuse of common parts/materials. The first crucial step of product recovery is disassembly.

Disassembly is defined as the methodical extraction of valuable parts/subassemblies and materials from discarded products through a series of operations. After disassembly, reusable parts/subassemblies are cleaned, refurbished, tested, and directed to the part/subassembly inventory for remanufacturing operations. The recyclable materials can be sold to raw-material suppliers, while the residuals are sent to landfills.

Disassembly has gained a great deal of attention in the literature recently due to its role in product recovery. A disassembly system faces many unique challenges; for example, it has significant inventory problems because of the disparity between the demands for certain parts or subassemblies and their yield from disassembly. The flow process is also different. As opposed to the normal "convergent" flow in regular assembly environment, in disassembly the flow process is "divergent" (a single product is broken down into many subassemblies and parts). There is also a high degree of uncertainty in the structure and the quality of the returned products. The conditions of the products received are usually unknown and the reliability of the components is suspected. In addition, some parts of the product may cause pollution or may be hazardous. These parts tend to have a higher chance of being damaged and hence may require special handling, which can also influence the utilization of the disassembly workstations. For example, an automobile slated for disassembly contains a variety of parts that are dangerous to remove or present a hazard to the environment such as the battery, airbags, fuel, and oil. Various demand sources may also lead to complications in disassembly line balancing. The reusability of parts creates a demand for them, however, the demands and availability of the reusable parts is significantly less predicable than what is found in the assembly process. Most products contain parts that are installed (and must be removed) in different attitudes, from different areas of the main structure, or in different directions. Since any required directional changes increase the setup time for the disassembly process, it is desirable to minimize the number of directional changes in the chosen disassembly sequence. Finally, disassembly line balancing is critical in minimizing the use of valuable resources (such as time and money) invested in disassembly and maximizing the level of automation of the disassembly process and the quality of the parts (or materials) recovered.

In this chapter, the disassembly line balancing problem (DLBP) is solved using exhaustive search and combinatorial optimization methodologies. Exhaustive search consistently provides the optimal solution, though its exponential time complexity quickly reduces its practicality. Combinatorial optimization techniques are instrumental in obtaining optimal or near-optimal solutions to problems with intractably large solution spaces. Combinatorial optimization is an emerging field that combines techniques from applied mathematics, operations research, and computer science to solve optimization problems over discrete structures. These techniques include: greedy algorithms, integer and linear programming, branch-and-bound, divide and conquer, dynamic programming, local optimization, simulated annealing, genetic algorithms (GAs), and approximation algorithms. In this chapter, an exhaustive search algorithm is presented for obtaining the optimal solution to small instances of the DLBP. A GA is then presented. The GA considered here involves a randomly generated initial population with crossover, mutation, and fitness competition performed over many generations. The DLBP is then solved using an ant colony optimization (ACO) metaheuristic. The ACO used here is an ant system (AS) algorithm known as the ant-cycle model [7] that is enhanced for DLBP. A deterministic hybrid process consisting of a greedy sorting algorithm followed by a hill-climbing heuristic (adjacent element hill climbing, AEHC) is then applied. Finally, a new general-purpose heuristic algorithm is presented for obtaining near-optimal solutions to the problem. Influenced by the hunter–killer (H–K) search tactics of military helicopters, this heuristic easily lends itself to the DLBP. All of these combinatorial optimization techniques seek to provide a feasible disassembly sequence, minimize the number of workstations, minimize the total idle time and minimize the variation in idle times between workstations while attempting to remove hazardous and high-demand product components as early as possible and remove parts with similar part removal directions together. Examples are considered to illustrate implementation of the methodologies. The conclusions drawn from the study include the consistent generation of optimal or near-optimal solutions, the ability to preserve precedence relationships, the superior speed of the methods, and their practicality due to the ease of implementation in solving DLBPs.

6.2 Literature Review

There are many steps involved in product recovery and remanufacturing. The first crucial step of product recovery is disassembly. Disassembly is a methodical extraction of valuable parts/subassemblies and materials from postused products through a series of operations. After disassembly, reusable parts/subassemblies are cleaned, refurbished, tested, and directed to the part/subassembly inventory for remanufacturing operations. The recyclable materials can be sold to raw-material suppliers and the residuals are disposed of. Many papers have discussed the different aspects of

product recovery. Brennan et al. [4] and Gupta and Taleb [20] investigated the problems associated with disassembly planning and scheduling. Torres et al. [49] reported a study for nondestructive automatic disassembly of personal computers. Gungor and Gupta [15,16,19] presented the first introduction to disassembly line balancing and developed an algorithm for solving the DLBP in the presence of failures with the goal of assigning tasks to workstations in a way that probabilistically minimizes the cost of defective parts [18]. For a review of environmental conscious manufacturing and product recovery, see Gungor and Gupta [17]. For a comprehensive review of disassembly sequencing, see Lambert [27] and Lambert and Gupta [28].

Gutjahr and Nemhauser [21] first described a solution to the assembly line balancing problem with an algorithm developed to minimize the delay times at each workstation. The heuristic accounts for precedence relationships and seeks to find the shortest path through a network with the resulting technique being similar to dynamic programming. Erel and Gokcen [11] developed a modified version of Gutjahr and Nemhauser's line balancing problem algorithm by allowing for mixed-model lines (assembly lines used to assemble different models of the same product) by allowing multiple state times then, during construction of the solution network, considering all state times before categorizing a given set of completed tasks to a workstation. Suresh et al. [48] first presented a GA to provide a near-optimal solution to the assembly line balancing problem that minimized idle time and minimized probability of line stoppage; that is, minimized the probability that the workstation time exceeds cycle time. Hackman et al. [22] proposed a branch-and-bound heuristic for the simple assembly line balancing type I problem, while Ponnambalam et al. [45] provided quantitative evaluations of various assembly line balancing techniques.

Dorigo et al. [9] first described ant colony optimization. Dorigo et al. [8] provide a summary and review of ant algorithms. McMullen and Tarasewich [42] used ant colony optimization techniques to solve the assembly line balancing problem with parallel workstations, stochastic task durations, and mixed models. Bautista and Pereira [2] used ant colony optimization techniques to solve the assembly line balancing problem and compared these to their hill climbing heuristics.

McGovern et al. [40] first proposed combinatorial optimization techniques for the DLBP. McGovern and Gupta demonstrated an early version of the H–K heuristic in [31].

6.3 Notation

The following notation is used in the remainder of the chapter:

$\leq_p$ Polynomial time reduction or polynomial transformation; read as: "can be converted to," "is easier than," "is a subset of," or "is a smaller problem than"

$\prec$	Partial ordering; that is, x precedes y is written $x \prec y$
$\langle 1, 2, \ldots, n \rangle$	Ordered n-tuple
$\{1, 2, \ldots, n\}$	Set (formal definition used here; i.e., listing of n distinct items)
$(1, 2, \ldots, n)$	List of n items
$\|X\|$	Cardinality of the set X
$\lceil x \rceil$	Ceiling function of x; assigns the smallest integer $\geq x$, for example, $\lceil 1.3 \rceil = 2$
$\forall$	"For all," "for every"
$\in$	"An element of," "is in"
$\exists$	"There exists," "there exists at least one," "for some"
O	"Big-O," $g(x)$ is $O(h(x))$ whenever $\exists\, a : \|g(x)\| \leq a\|h(x)\| \forall\, x \geq 0$
!	Factorial
:	"Such that"
$\Delta\psi_k$	kth Element's delta skip measure; difference between problem size, n, and skip size, ψ_k (i.e., for $\Delta\psi = 10$ and $n = 80$, $\psi = 70$)
ψ_k	kth Element's skip measure (i.e., for the solution's third element, visit every second possible task, if $\psi_3 = 2$)
α	Weight of existing pheromone (trail) in path selection
β	Weight of the edges in path selection
η_{pq}	Visibility value of edge (arc for DLBP) pq at time t
ρ	Variable such that $1 - \rho$ represents the pheromone evaporation rate
τ_{pq}(NC)	Amount of trail on edge pq (arc for DLBP) during cycle NC
a	Multiprocessor scheduling problem task variable
A	Multiprocessor scheduling problem task set
B	Function variable in complexity theory
B	Multiprocessor scheduling problem deadline bound
BST_k	kth Part in temporary best solution sequence during AEHC
c	Initial amount of pheromone on all of the paths at time $t = 0$
CT	Cycle time; maximum time available at each workstation
d_k	Demand; quantity of part k requested
D	Demand rating for a given solution sequence; also demand bound for the decision version of DLBP
D^*	Lower demand bound (optimal) for a given instance; also used to refer to the set of solutions optimal in D
D_{nom}	Upper demand bound (nominal) for a given instance
DP	The set of demanded parts
F	Measure of balance for a given solution sequence; fitness measure in GA
F^*	Lower measure of balance bound (optimal) for a given instance; also used to refer to the set of solutions optimal in F
F_{nom}	Upper measure of balance bound (nominal) for a given instance
F_{tr}	Measure of balance of ant r's sequence at time t
FS	Feasible sequence binary value; FS = 1 if feasible, 0 otherwise
h_k	Binary value; 1 if part k is hazardous, else 0

H	Hazard rating for a solution; also hazard bound for the decision version of DLBP
H^*	Lower hazard bound (optimal) for a given instance; also used to refer to the set of solutions optimal in H
H_{nom}	Upper hazard bound (nominal) for a given instance
HP	Set of hazardous parts
i	Counter variable
I	Total idle time for a given solution sequence; also refers to an instance of a problem in complexity theory
I^*	Lower idle time bound (optimum) for a given instance
I_j	Total idle time of workstation j
I_{nom}	Upper idle time bound (nominal) for a given instance
ISS_k	Binary value; 1 if kth part is in solution sequence, else 0
j	Workstation count (1, …, NWS)
k	Part identification (1, …, n)
$l(a)$	Multiprocessor scheduling problem task length
L	A given problem in complexity theory
L_r	ACO delta trail divisor value; set equal to F_{nr} for DLBP
m	Number of processors in the multiprocessor scheduling problem; also number of ants
n	Number of parts for removal
N	Number of chromosomes (population)
N	Set of natural numbers; that is, {0, 1, 2, . . .}
NC_{max}	Maximum number of cycles for ACO
NPW_j	Number of parts in workstation j
NWS	Number of workstations required for a given solution sequence
NWS*	Minimum possible number of workstations for n parts
NWS_{nom}	Maximum possible number of workstations for n parts
p	Counter variable; also edge/arc variable providing node/vertex identification
$p^r_{pq}(t)$	Probability of ant r taking an edge (arc for DLBP) pq at time t during cycle NC
P	Set of n part removal tasks
PRT	Set of part removal times
PRT_k	Part removal time required for kth part
PS_k	kth Part in a solution sequence; that is, for solution ⟨3, 1, 2⟩, $PS_2 = 1$
PSG	Solution sequence after application of the greedy algorithm
PSS_k	kth Part in solution sequence after sorting
PST	Solution sequence after AEHC
q	Counter variable; also edge/arc variable (node/vertex identification)
Q	Amount of pheromone added if a path is selected
r	Set of unique part removal directions; also ant count
r_k	Integer value corresponding to the kth part's removal direction

R	Direction rating for a given solution sequence; also direction bound for the decision version of DLBP
R^*	Lower direction bound (optimal) for a given instance; also used to refer to the set of solutions optimal in R
R_k	Binary value; 0 if part k can be removed in the same direction as part $k + 1$, else 1
R_m	Mutation rate
R_{nom}	Upper direction bound (nominal) for a given instance
R_x	Crossover rate
ST_j	Station time; total processing time requirement in workstation j
t	Time within a cycle; ranges from 0 to n
TMP_k	kth Part in temporary sequence during AEHC
V	Maximum range for a workstation's idle time
x	General variable; also refers to a part removal direction
y	General variable; also refers to a part removal direction
z	General variable; also refers to a part removal direction
Z	Set of integers; that is, { …, −2, −1, 0, 1, 2, …}
Z^+	Set of positive integers; that is, {1, 2, …}
Z_p	pth Objective to minimize or maximize

6.4 Considerations Related to Disassembly Lines

Gungor and Gupta [19] first formally proposed the disassembly line in 2002. The following is a summary of some of the various considerations in a disassembly line setting due to their observation of a disassembly line as being fraught with a variety of disassembly-unique complications. The following is a reprinted listing and description of these considerations.

6.4.1 Product Considerations

The number of different products disassembled on the same line is an important characteristic of a disassembly line. The line may deal with only one type of product whose original configuration is the same for every product received; for example, only Pentium-based PCs with certain specifications. The line may also be utilized to disassemble products belonging to the same family; that is, whose original configurations are only slightly different from each other. For example, we may have different models of PCs, such as Pentium-based PCs, 486-based PCs, etc. on the same line. The line may receive several types of products, partially disassembled products and subassemblies whose configurations are significantly or completely different from one another such as PCs, printers, digital cameras, motherboards, and monitors.

Changing characteristics of the products complicates the disassembly operations on the line. Intuitively, balancing a disassembly line used for the

disassembly of several types of products can become very complex. Such a line may be balanced for a group of products, yet its status may become largely or completely unbalanced when a new type of product is received.

6.4.2 Line Considerations

Various disassembly line configurations may be possible. Some layouts will probably be inspired by assembly lines layouts. For example, layouts like serial, parallel, circular, U-shaped, cellular, and two-sided lines [47] may also find their way onto disassembly lines. Even so, new layouts may still be required to create more efficient disassembly lines.

One of the most important considerations of a disassembly line is the line speed. Disassembly lines could be either paced or unpaced. Paced lines may be used to regulate the flow of parts on disassembly lines. However, if there is too much variability in the task times (which might depend on the conditions of the products being disassembled or the variety of the products processed on the line), it might be preferable to have an unpaced line. In such lines, each station works at its own pace and advances the part to the next station whenever it completes the assigned tasks. The advantages of paced lines over unpaced lines include considerably less work in process, less space requirements, more predictable and higher throughput and, if properly handled, less chance of bottlenecks. To take advantage of the positive aspects of a paced line, its speed can be dynamically modified throughout the entire disassembly process to minimize the negative effects of variability (including variability in demand).

6.4.3 Part Considerations

6.4.3.1 Quality of Incoming Products

When a disassembly system receives the returned products, their conditions are usually unknown; sometimes they are in good shape and relatively new, while at other times they are old, nonfunctioning items. Therefore, there is a high level of uncertainty in the quality of the products and their constituent parts. There is a trade-off between the level of uncertainty and the efficiency of the disassembly line. When the level of uncertainty increases, the efficiency measures worsen, especially if the disassembly line has not been designed to cope with the uncertainty factors.

A part is considered to be *defective* if it has a different structure or different operational specifications than its original structure and specifications. A part can become defective in various ways, such as being in an accident or being exposed to a hostile operating environment. There are mainly two types of defects:

- *Physical defect*. A part can be physically defective indicating that the geometric specifications (dimensions and shape) of the part are

different than its original design. For example, the CRT of a computer monitor can be broken, or the cover of a PC can be dented. AQ1

- *Functional defect*. A part may not contain any physical defect; however, it may not function as it was originally designed to. For example, the CPU of a PC may be perfect in its physical aspects but it may not function due to an internal problem.

6.4.3.1.1 Removable Defective Parts (Type A Defect)

Some of the physically defective parts in the product can be disassembled even though they sustain some level of damage. We call this type of physical defect a "type A defect." The precedence relationships must be satisfied during the assignment of the disassembly tasks to the disassembly workstations. When a part is physically defective and yet removable, it might take longer to disassemble that part. However, it does not affect the disassembly of other parts since the precedence relationship is not affected by the longer disassembly time.

6.4.3.1.2 Nonremovable Defective Parts (Type B Defect)

Sometimes a physically defective part in the product cannot be disassembled because it is either badly damaged (and thus gets stuck in its place) or its fixture is of the type that cannot be undone. This is a "type B defect" and it has a tremendous impact on the efficiency of the disassembly line. For example, even if the nonremovable part does not have a demand, it may still precede other demanded part(s). This case may have various levels of negative effects on the disassembly line depending on the number of demanded (and not necessarily all) parts that are preceded by the nonremovable part. A part with a type B defect may also result in what we call the *disappearing workpieces phenomena* (DWP) (see Section 6.4.4.5 for a discussion on DWP).

6.4.3.1.3 Parts with Functional Defects

Assume that part k does not have a physical defect (Types A or B), but rather sustains a functional defect. If this fact were known in advance, then the disassembler may or may not have an incentive to disassemble it (unless, of course, it precedes other demanded parts). Therefore, in addition to concerns related to physical defects on parts, the possibility of functional defects on parts in incoming products should also be incorporated in operating and balancing a disassembly line.

6.4.3.2 Quantity of Parts in Incoming Products

Another complication is related to the quantity of parts in the incoming products. Due to upgrading (or downgrading) of the product during its use, the number of parts in the product may not match with its original configuration.

The actual number of parts may be more or less than expected when the product is received.

The following should be considered:

- The demanded part of the product may be absent or its quantity may be less than expected. This may require a provision in the calculation to ensure that the demand for the missing parts is met. Another approach would be to carry out a preliminary evaluation of the product to make sure that all the parts of interest exist in the product before it is pushed down on the disassembly line. This evaluation may be used to determine the route of the product on the disassembly line.
- The number of demanded parts may be more than expected. The most common example would be the example of a PC disassembly system. Assume that there is a demand for the memory modules of old PCs. Further assume that PCs being disassembled were originally configured to have one 32 MB memory module. However, if the owner of the PC upgraded the computer's memory to 64 MB using another 32 MB module, two 32 MB memory modules would be retrieved as a result of the disassembly of the PC, which is one more than expected. This situation would affect the memory demand constraints and the workstation time (since the disassembly of two memory modules would take longer than the removal of just one).

6.4.4 Operational Considerations

6.4.4.1 Variability of Disassembly Task Times

Similar to the assembly line case, the disassembly task times may vary depending on several factors that are related to the condition of the product and the state of the disassembly workstation (or worker). Task times may be considered as deterministic, stochastic, or dynamic. Dynamic task times are possible due to learning effects, which allow systematic reduction in disassembly times. For example, in their case based reasoning (CBR) approach, Zeid et al. [51] demonstrated that the disassembly of products can be assisted by the reuse of solutions to similar problems encountered in the past, thus reducing disassembly times.

6.4.4.2 Early Leaving Workpieces

If one or more (not all) tasks of a workpiece that has been assigned to the current workstation cannot be completed due to some defect, the workpiece might leave the workstation early. This phenomenon is termed as the *early leaving workpiece* (EWP).

Due to EWP, the workstation experiences an unscheduled idle time for the duration of the tasks that causes the workpiece to leave early. Note that

the cost of unscheduled idle time is high because, although the demand for parts associated with failed tasks is unfulfilled, the disassembly cost of failed tasks has been incurred. For example, assume that disassembly of the hard disk and the floppy drive of a PC is carried out at workstation 1. After the removal of the hard disk, if the holding screws of the floppy drive are stuck or damaged, we may not be able to remove the floppy. Therefore, the workpiece (the PC being disassembled) can leave workstation 1 early. When this happens, workstation 1 remains idle for the duration of the normal disassembly time of floppy.

6.4.4.3 Self-Skipping Workpieces

If all tasks of a workpiece that have been assigned to the current workstation are disabled due to some defect they posses or precedence relationships, the workpiece leaves the workstation early without being worked on. This phenomenon is known as *self-skipping workpiece* (SSWP).

Consider the disassembly of the PC example given above. Assume that the disassembly of the hard disk must be carried out before the disassembly of the floppy drive. If the hard disk of the PC cannot be removed because its holding screws are damaged, neither of the tasks assigned to workstation 1 (disassembly of the hard disk and the floppy drive) can be done. Therefore, the workpiece can leave the workstation without being worked on; in other words, it self-skips workstation 1.

6.4.4.4 Skipping Workpieces

At workstation j, if one or more defective tasks of a workpiece directly or indirectly precede *all* the tasks of workstation $j + 1$ (i.e., the workstation immediately succeeding workstation j), the workpiece "skips" workstation $j + 1$ and moves on to workstation $j + 2$. This phenomenon is known as a *skipping workpiece* (SWP).

The number of workstations a work-piece skips is the strength of skipping. If a workpiece skips only one workstation, it is a 1-SWP; if there are two workstations skipped, it is a 2-SWP, and so on. In addition to unscheduled idle time, both SSWP and SWP experience added complexities in material handling (e.g., how to transfer the workpiece out of turn to the downstream workstation) and the status of the downstream workstation (e.g., it may be busy working on other workpieces and may require some sort of buffer allocation procedure to hold the skipped workpiece until the machine becomes available). Using the disassembly of a PC given before, assume the disassembly of the floppy drive precedes the removal of the RAM modules and the sound card of the PC scheduled to be carried out at workstation 2. If the floppy drive cannot be removed at workstation 1 due to its damaged holding screws, the workpiece leaves workstation 1 early and may skip workstation 2 because neither the RAM modules nor the sound card can be removed from the workpiece. In this case, workstation 2 remains idle for the

duration of its originally scheduled tasks, namely the removal of the RAM modules and the sound card.

6.4.4.5 Disappearing Workpieces

If a defective task disables the completion of all the remaining tasks on a workpiece, the workpiece may simply be taken off the disassembly line before it reaches any downstream workstation. In other words, the workpiece effectively disappears, which is referred to as a *disappearing workpiece* (DWP).

DWP may result in starvation of subsequent workstations leading to a higher overall idle time, highly undesirable in a balanced line. DWP, in a way, is a special case of SWP, where the workpiece skips all succeeding workstations. The consequences of the DWP are similar to the SWP, but to a greater extent.

For example, assume that the disassembly of the top cover of the PC is carried out at workstation 1 in addition to the disassembly of hard disk and the floppy drive. If the top cover cannot be removed due to damage, the PC (workpiece) can be taken off the line since we cannot reach any of the parts inside the PC. In a way, it disappears and the idle times propagate throughout the remaining workstations.

6.4.4.6 Revisiting Workpieces

A workpiece currently at workstation j may *revisit* a preceding workstation $(j - p)$, where $(j - p) \geq 1$ and $p \geq 1$ and integer, to perform task x if the completion of current task y enables one to work on task x, which was originally assigned to workstation $(j - p)$ and was, however, disabled due to the failure of another preceding task. These are termed *revisiting workpieces* (RWP).

An RWP results in overloading one of the previous workstations. As a result, it may lead to complications in the material-handling system because of reverse flows, decoupling from the revisited workstation or introduction of a buffer to hold the revisiting workpiece until the workstation becomes available. This would obviously have financial consequences as well.

To exemplify RWP, assume that at workstation 1, the hard disk and the floppy drive are disassembled; at workstation 2, the RAM modules and the sound card are disassembled; and at workstation 3 the power unit of a PC is disassembled. Also assume that RAM modules can be removed after the removal of either the floppy drive or the power unit. Further assume that disassembly of the floppy drive precedes the removal of the sound card. If the floppy drive cannot be removed due to preexisting damage, the PC skips workstation 2 and goes to workstation 3 where the power unit is removed. Since the power unit has been taken out of the PC, the RAM modules can be removed. Thus, the PC (workpiece) is sent back to the workstation 2 for the disassembly of the RAM modules. Complicated RWPs that could make the material handling and flow control more difficult may also exit the line.

6.4.4.7 Exploding Workpieces

A workpiece may split into two or more workpieces (subassemblies) as it moves on the disassembly line because of the disassembly of certain parts that hold the workpiece together. Each of these subassemblies acts as an individual workpiece on the disassembly line. This phenomenon is known as *exploding workpieces* (EWP).

AQ2

EWP complicates the flow mechanism of the disassembly line, however EWP can be planned in advance since we know which part's removal would result in the EWP. In a disassembly line, a complication occurs when the part that would normally result in EWP cannot be removed due to some defect.

6.4.5 Demand Considerations

Demand is one of the most crucial issues in disassembly line design and optimization since we wish to maximize the utilization of the disassembly line while meeting the demand for parts in associated planning periods. In disassembly, the following demand scenarios are possible: demand for one part only (single part disassembly—a special case of partial disassembly); demand for multiple parts (partial disassembly); and demand for all parts (complete disassembly).

Parts with physical defects or with functional defects may influence the performance of the disassembly line. If part x is not demanded and it directly or indirectly precedes a demanded part y, then part x must be disassembled before the removal of part y. (Note that when part a is precedent to part b, which is precedent to part c, then part a is said to be a *direct precedent* to part b and an *indirect precedent* to part c.) Removal of part x may require additional time due to the presence of a defect, which, for example, may extend the access time. This may cause the processing time to exceed the cycle time, which could result in the starvation of subsequent workstations and blockage of the previous workstations. Placing buffers at the workstations may be necessary to avoid starvation, blockage, and incomplete disassembly. If part x has a demand, then the various types of demand sources may lead to complications. These demand considerations affect the number of products to be disassembled, and eventually the disassembly line balance. There are three types of demand.

The first type is as follows: the demand source may accept part k "as is." This is possible, for example, when part k is demanded for its material content (in which case the defect in the part may not be important).

The second-type demand is described as follows. The demand source may not accept parts with any type of defect. This occurs when a part is used "as is" (e.g., in remanufacturing or repairs of other products). Thus, if a demanded part has a defect, its disassembly does not satisfy the demand. Since the primary objective of a disassembly line is to meet the demand, the objective function and demand constraints must cope with this type of

complication. If the part has second-type demand and it does not directly or indirectly precede another demanded part, the disassembly of the part is redundant. This may have an effect on the efficiency of the disassembly line since the workstation responsible for removing the defective part will remain idle for some time.

In the third-type demand, the demand source may accept certain defective parts depending on the seriousness of the defect. This type of demand may be received from a refurbishing environment where the parts undergo several correction processes (such as cleaning and repair) before reuse. This type of demand introduces a further complication, which requires some sort of tracing mechanism to identify the type of defect and the associated demand constraint.

6.4.6 Assignment Considerations

In addition to precedence relationships, several other restrictions limit the assignment of tasks to workstations. Although similar restrictions are also present in assembly lines, the following are some restrictions related specifically to disassembly.

Certain tasks must be grouped and assigned to a specific workstation for a reason. For example, the removed parts will be sorted and packaged together for shipment to the demand source. Thus, assigning the disassembly of these components (parts or joining elements) to the same workstation minimizes the distance that the components travel in the disassembly system. Moreover, the tasks requiring similar operating conditions (e.g., temperature and lighting) can be restricted to certain workstations as well.

The availability of special machining and tooling at certain workstations may necessitate the assignment of certain tasks to these workstations. For example, disassembly of hazardous parts may be assigned to highly specialized workstations to minimize the possibility of contamination in the rest of the system. Similarly, the skills of human workers can be a factor in the task assignment restrictions.

Tasks may be assigned to workstations such that the amount of repositioning of the workpieces on the disassembly line (e.g., the disassembly direction changes) is minimized. Similarly, tasks may have to be assigned to minimize the number of tool changes throughout the disassembly process.

6.4.7 Other Considerations

There are additional uncertainty factors associated with the reliability of the disassembly workstations. Some parts of the product may cause pollution or nuisance due to the nature of their contents (e.g., oil and gas), which may increase the chance of breakdowns or workstation downtime. Furthermore, hazardous parts may require special handling, which can also influence the utilization of the workstations.

6.5 DLBP Model Considerations

The DLBP investigated in this chapter is concerned with a paced disassembly line for a single model of product that undergoes complete disassembly. Part removal times are known and discrete; hence they do not posses any probabilistic component. A summary of model assumptions is listed at the end of this section.

The desired solution to a DLBP instance consists of an ordered sequence (n-tuple) of work elements (also referred to as tasks, components, or parts). For example, if a solution consisted of the 8-tuple $\langle 5, 2, 8, 1, 4, 7, 6, 3\rangle$, then component 5 would be removed first, followed by component 2, then component 8, and so on.

While different authors use a variety of definitions for the term "balanced" in reference to assembly [10] and disassembly lines, we propose the following definition [40] [30] that will be used consistently throughout this chapter:

> ***Definition*** A disassembly line is optimally *balanced* when the fewest possible number of workstations is needed and the variation in idle times between all workstations is minimized. This is mathematically described by
>
> $$\text{Minimize NWS}$$
>
> then
>
> $$\text{Minimize } [\max(ST_x) - \min(ST_y)] \quad \forall x, y \in \{1, 2, \ldots, \text{NWS}\}$$

Except for some minor variation in the greedy/hill-climbing approach, all of the combinatorial optimization techniques described here use a similar methodology to address the multicriteria aspects of DLBP. Since measure of balance is the primary consideration in this chapter, additional objectives are only considered subsequently; that is, the methodologies first seek to select the best performing measure of balance solution as given by Formula 6.4; equal balance solutions are then evaluated for hazardous part removal positions per Formula 6.13; equal balance and hazard measure solutions are evaluated for high-demand part removal positions as measured by Formula 6.21; and equal balance, hazard measure, and high-demand part removal position solutions are evaluated for the number of direction changes as measured by Formula 6.26. Based in lexicographic goal programming, this priority ranking approach was selected over a weighting scheme for its simplicity, ease in reranking the priorities, ease in expanding or reducing the number of priorities, due to the fact that other weighting methods can be readily addressed at a later time, and primarily to enable unencumbered efficacy analysis of the combinatorial optimization methodologies and problem data instances under consideration.

Model assumptions include the following:

- The line is paced.
- Part removal times are deterministic, constant, and discrete (or able to be converted to integer form).
- Each product undergoes complete disassembly (even if demands are zero).
- All products contain all parts with no additions, deletions, modifications, or physical defects.
- Each part is assigned to one and only one workstation.
- The sum of the part removal times of all the parts assigned to a workstation must not exceed CT.
- The precedence relationships among the parts must not be violated.

6.6 The DLBP Model Mathematical Description

The particular problem investigated in this chapter seeks to fulfill five objectives:

1. Minimize the number of disassembly workstations and hence, minimize the total idle time
2. Ensure the idle times at each workstation are similar
3. Remove hazardous components early in the disassembly sequence
4. Remove high-demand components before low-demand components
5. Minimize the number of direction changes required for disassembly

A major constraint is the requirement to provide a feasible (i.e., precedence preserving) disassembly sequence for the product being investigated. The result is an integer, deterministic, *n*-dimensional, multiple-criteria decision-making problem with an exponential search space. Testing a given solution against the precedence constraints fulfills the major constraint of precedence preservation. Minimizing the sum of the workstation idle times, which will also minimize the total number of workstations, attains objective 1 and is described by

$$I = (\mathrm{NWS} \cdot \mathrm{CT}) - \sum_{k=1}^{n} \mathrm{PRT}_k \tag{6.1}$$

or

$$I = \sum_{j=1}^{\mathrm{NWS}} (\mathrm{CT} - \mathrm{ST}_j) \tag{6.2}$$

This objective is represented as

$$\text{Minimize } Z_1 = \sum_{j=1}^{\text{NWS}} (\text{CT} - \text{ST}_j) \tag{6.3}$$

Line balancing seeks to achieve *perfect balance* (all idle times equal to zero). When this is not achievable, either line efficiency (LE) or the smoothness index (SI) is used as a performance evaluation tool [10]. We use a measure of balance that combines the two and is easier to calculate. SI rewards similar idle times at each workstation, but at the expense of allowing for a large (suboptimal) number of workstations. This is because SI compares workstation elapsed times to the largest ST_j instead of the CT. LE rewards the minimum number of workstations, but allows unlimited variance in idle times between workstations because no comparison is made between ST_js. This chapter makes use of the balancing method developed by McGovern and Gupta [30,40]. The McGovern–Gupta balancing method simultaneously minimizes the number of workstations while aggressively ensuring that idle times at each workstation are similar, though at the expense of the generation of a nonlinear objective function. The method is computed based on the minimum number of workstations required, as well as the sum of the square of the idle times for each of the workstations. This penalizes solutions where, although the number of workstations may be minimized, one or more have an exorbitant amount of idle time when compared to the other workstations. It provides for leveling the workload between different workstations on the disassembly line. Therefore, a resulting minimum numerical performance value is the more desirable solution indicating both a minimum number of workstations and similar idle times across all workstations. The McGovern–Gupta measure of balance is represented as

$$F = \sum_{j=1}^{\text{NWS}} (\text{CT} - \text{ST}_j)^2 \tag{6.4}$$

with the DLBP balancing objective represented as

$$\text{Minimize } Z_2 = \sum_{j=1}^{\text{NWS}} (\text{CT} - \text{ST}_j)^2 \tag{6.5}$$

Perfect balance is indicated by

$$Z_2 = 0 \tag{6.6}$$

Note that mathematically, Formula 6.5 effectively makes Formula 6.3 redundant due to the fact that it concurrently minimizes the number of workstations.

Theorem 6.1 *Let* PRT_k *be the part removal time for the kth of n parts, where CT is the maximum amount of time available to complete all tasks assigned to each workstation. Then for the most efficient distribution of tasks, the minimum number of workstations, NWS* satisfies*

$$\mathrm{NWS}^* \geq \left\lceil \frac{\sum_{k=1}^{n} \mathrm{PRT}_k}{\mathrm{CT}} \right\rceil \tag{6.7}$$

***Proof*:** If the above inequality is not satisfied, then there must be at least one workstation completing tasks requiring more than *CT* of time, which is a contradiction.

Subsequent bounds are shown to be true in a similar fashion and are presented throughout the chapter without proof.

The upper bound for the number of workstations is given by

$$\mathrm{NWS}_{\mathrm{nom}} = n \tag{6.8}$$

Therefore

$$\left\lceil \frac{\sum_{k=1}^{n} \mathrm{PRT}_k}{\mathrm{CT}} \right\rceil \leq \mathrm{NWS} \leq n \tag{6.9}$$

The lower bound on *F* is given by

$$F^* \geq \left(\frac{I}{\mathrm{NWS}^*}\right)^2 \cdot \mathrm{NWS}^* \tag{6.10}$$

while the upper bound is described by

$$F_{\mathrm{nom}} = \sum_{k=1}^{n} (\mathrm{CT} - \mathrm{PRT}_k)^2 \tag{6.11}$$

therefore

$$\left(\frac{I}{\mathrm{NWS}^*}\right)^2 \cdot \mathrm{NWS}^* \leq F \leq \sum_{k=1}^{n} (\mathrm{CT} - \mathrm{PRT}_k)^2 \tag{6.12}$$

A hazard measure was developed to quantify each solution sequence's performance, with a lower calculated value being more desirable [33]. This measure is based on binary variables that indicate whether a part is considered to contain hazardous material (the binary variable is equal to one if the part is hazardous, else zero) and its position in the sequence. A given solution sequence hazard measure is defined as the sum of hazard binary flags multiplied by their position in the solution sequence, thereby rewarding the removal of hazardous parts early in the part removal sequence. This measure is represented as

$$H = \sum_{k=1}^{n} \left(k \cdot h_{PS_k}\right) \quad h_{PS_k} = \begin{cases} 1, \text{ hazardous} \\ 0, \text{ otherwise} \end{cases} \tag{6.13}$$

with the DLBP hazardous part objective represented as

$$\text{Minimize } Z_3 = \sum_{k=1}^{n} \left(k \cdot h_{PS_k}\right) \tag{6.14}$$

The lower bound on the hazardous part measure is given by

$$H^* = \sum_{p=1}^{|\text{HP}|} p \tag{6.15}$$

where the set of hazardous parts is defined as

$$\text{HP} = \{k : h_k \neq 0 \; \forall \; k \in P\} \tag{6.16}$$

and its cardinality can be calculated with

$$|\text{HP}| = \sum_{k=1}^{n} h_k \tag{6.17}$$

For example, a product with three hazardous parts would give an H^* value of $1 + 2 + 3 = 6$. The upper bound on the hazardous part measure is given by

$$H_{\text{nom}} = \sum_{p=n-|\text{HP}|+1}^{n} p \tag{6.18}$$

or alternatively

$$H_{\text{nom}} = (n \cdot \; |\text{HP}|) - \; |\text{HP}| \tag{6.19}$$

For example, three hazardous parts in a product having a total of 20 would give an H_{nom} value of $18 + 19 + 20 = 57$ or equivalently, $H_{\text{nom}} = (20 \cdot 3) - 3 = 60 - 3 = 57$. Formulae 6.15, 6.18, and 6.19 are combined to give

$$\sum_{p=1}^{|\text{HP}|} p \leq \text{H} \leq \sum_{p=n-|\text{HP}|+1}^{n} p = (n \cdot \; |\text{HP}|) - \; |\text{HP}| \tag{6.20}$$

Also, a demand measure was developed to quantify each solution sequence's performance, with a lower calculated value being more desirable [33]. This measure is based on positive integer values that indicate the quantity required of this part after it is removed—or zero if it is not desired—and its position in the sequence. Any given solution sequence demand measure is defined as the sum of the demand value multiplied by their position in the sequence, rewarding the removal of high-demand parts early in the part removal sequence. This measure is represented as

$$D = \sum_{k=1}^{n} \left(k \cdot d_{\text{PS}_k}\right) \quad d_{\text{PS}_k} \in \text{N}, \forall \text{PS}_k \tag{6.21}$$

with the DLBP demand part objective represented as

$$\text{Minimize } Z_4 = \sum_{k=1}^{n} \left(k \cdot d_{\text{PS}_k}\right) \tag{6.22}$$

The lower bound on the demand measure (D^*) is given by Formula 6.21, where

$$d_{\text{PS}_1} \geq d_{\text{PS}_2} \geq \cdots \geq d_{\text{PS}_n} \tag{6.23}$$

For example, three parts with demands of 4, 5, and 6, respectively would give a best-case value of $(1 \cdot 6) + (2 \cdot 5) + (3 \cdot 4) = 28$. The upper bound on the demand measure (D_{nom}) is given by Formula 6.21, where

$$d_{\text{PS}_1} \leq d_{\text{PS}_2} \leq \cdots \leq d_{\text{PS}_n} \tag{6.24}$$

For example, three parts with demands of 4, 5, and 6 respectively would give a worst-case value of $(1 \cdot 4) + (2 \cdot 5) + (3 \cdot 6) = 32$.

Finally, a direction measure was developed to quantify each solution sequence's performance, with a lower calculated value indicating minimal direction changes and a more desirable solution [35]. This measure is based on a count of the direction changes. Integer values represent each possible direction (typically $r = \{+x, -x, +y, -y, +z, -z\}$; in this case $|r| = 6$). These directions are expressed as

$$r_{PS_k} = \begin{cases} +1, & \text{direction} + x \\ -1, & \text{direction} - x \\ +2, & \text{direction} + y \\ -2, & \text{direction} - y \\ +3, & \text{direction} + z \\ -3, & \text{direction} - z \end{cases} \tag{6.25}$$

and are easily expanded to other or different directions in a similar manner. The direction measure is represented as

$$R = \sum_{k=1}^{n-1} R_k \qquad R_k = \begin{cases} 1, & r_{PS_k} \neq r_{PS_{k+1}} \\ 0, & \text{otherwise} \end{cases} \tag{6.26}$$

with the DLBP direction part objective represented as

$$\text{Minimize } Z_5 = \sum_{k=1}^{n-1} R_k \tag{6.27}$$

The lower bound on the direction measure is given by

$$R^* = |r| - 1 \tag{6.28}$$

For example, for a given product containing six parts that are installed/removed in directions $r_k = (-y, +x, -y, -y, +x, +x)$ the resulting best-case value would be $2 - 1 = 1$ (e.g., one possible R^* solution containing the optimal, single-change of product direction would be: $\langle -y, -y, -y, +x, +x, +x \rangle$). In the specific case, where the number of unique direction changes is one less than the total number of parts n, the upper bound on the direction measure would be given by

$$R_{\text{nom}} = |r| \quad \text{where } |r| = n - 1 \tag{6.29}$$

Otherwise, the measure varies depending on the number of parts having a given removal direction and the total number of removal directions. It is bounded by

$$|r| \leq R_{\text{nom}} \leq n - 1 \quad \text{where } |r| < n - 1 \tag{6.30}$$

For example, six parts installed/removed in directions $r_k = (+x, +x, +x, -y, +x, +x)$ would give an R_{nom} value of 2 as given by the lower bound of Formula 6.30 with a solution sequence of $\langle +x, +x, -y, +x, +x, +x \rangle$. Six parts installed/removed in directions $r_k = (-y, +x, -y, -y, +x, +x)$ would give an R_{nom} value of $6 - 1 = 5$ as given by the upper bound of Formula 6.30 with a solution sequence of $\langle -y, +x, -y, +x, -y, +x \rangle$, for example.

In the special case, where each part has a unique removal direction, the measures for R^* and R_{nom} are equal and are given by

$$R^* = R_{\text{nom}} = n - 1 \quad \text{where } |r| = n \tag{6.31}$$

Note that the preceding optimal and nominal hazard, demand, and direction formulae are dependent on favorable precedence constraints that will allow for generation of these optimal or nominal measures.

6.7 Computational Complexity of DLBP

Although similar in name to the assembly line balancing problem (a claimed NP-hard combinatorial optimization problem), the DLBP contains a variety of complexity-increasing enhancements. The general assembly line balancing

problem can be formulated as a scheduling problem, specifically as a flow shop [44]. In its most basic form, the objective of DLBP is to minimize the idle times found at each machine (workstation). Since finding the optimal solution requires investigating all permutations of the sequence of n part removal times, there are $n!$ possible solutions. The time complexity of searching this space is $O(n!)$, referred to as *exponential time*. Though a possible solution to an instance of this problem can be verified in polynomial time, it cannot always be optimally solved. Therefore this problem is classified as nondeterministic polynomial (NP); if the problem could be solved in polynomial time, it would then be classified as polynomial (P). (Note that a great deal of the explanations that follow can be found in Garey and Johnson [12] and Tovey [50].)

NP-completeness provides strong evidence that the problem cannot be solved with a polynomial time algorithm (the theory of NP-completeness is applied only to decision problems). Many problems that are polynomial solvable differ only slightly from other problems that are NP-complete. 3-Satisfiability and 3-dimensional matching are NP-complete while the related 2-satisfiability and 2-dimensional matching problems can be solved in polynomial time (polynomial time algorithm design is detailed in Refs. 1 and 46). If a problem is NP-complete, some subproblem (created by additional restrictions being placed on the allowed instances) may be solvable in polynomial time. Certain encoding schemes and mathematical techniques can be used on some size-limited cases of NP-complete problems (an example is a dynamic programming approach to partition that results in a search space size equal to the size of the instance multiplied by the log of the sum of each number in the instance) enabling the solution by what is know as a "pseudopolynomial time algorithm." In the PARTITION case, this results in a polynomial time solution as long as extremely large input numbers are not found in the instance. Problems for which no pseudopolynomial time algorithm exists (assuming P $\neq$ NP) are known as "NP-complete in the strong sense" (also, *unary* NP-complete—referring to allowance for a nonconcise or unary encoding scheme notation where a string of n 1s represents the number n—with an alternative being *binary* NP-complete). Typical limiting factors for NP-complete problems to have a pseudopolynomial time algorithm include the requirement that the problem be a number problem (versus a graph problem, logic problem, etc.) and be size constrained, typically in the number of instance elements (n in DLBP) and the size of the instance elements (PRT_k in DLBP). Formally, an algorithm that solves a problem L will be called a *pseudopolynomial time algorithm* for L, if its time complexity function is bounded above as a function of the instance I by a polynomial function of the two variables: Length[I] and Max[I]. A general NP-completeness result implies that the problem cannot be solved in polynomial time in all the chosen parameters.

Knapsack has been shown to be solvable by a pseudopolynomial time algorithm in a dynamic programming fashion similar to partition by Dantzig [6]. Multiprocessor scheduling and sequencing within intervals have both been

shown to be NP-complete in the strong sense. The same is true for 3-partition, which is similar to partition but has m (instead of 2) disjoint sets and each of the disjoint sets has exactly three elements. By considering subproblems created by placing restrictions on one or more of the natural problem parameters, useful information about what types of algorithms are possible for the general problem can be gleaned.

The decision version of DLBP is NP-complete in the strong sense. This can be shown through a proof by restriction to multiprocessor scheduling. Garey and Johnson [12] describe multiprocessor scheduling as:

> *Instance*: A finite set A of tasks, a length $l(a) \in Z^+$ for each $a \in A$, a number $m \in Z^+$ of processors, and a deadline $B \in Z^+$.
>
> *Question*: Is there a partition $A = A_1 \cup A_2 \cup \cdots \cup A_m$ of A into m disjoint sets, such that
>
> $$\max \sum_{a \in A_i} l(a) : 1 \leq i \leq m \leq B?$$

In DLBP, NWS is equivalent to m in multiprocessor scheduling, P is equivalent to A, while CT is equivalent to B.

The following theorem provides the proof:

> ***Theorem 6.2*** DLBP is NP-complete in the strong sense.
>
> *Instance*: A finite set P of tasks, partial order $\prec$ on P, task time $\text{PRT}_k \in Z^+$, hazardous part binary value $h_k \in \{0, 1\}$, part demand $d_k \in \mathbf{N}$, and part removal direction $r_k \in Z$ for each $k \in P$, workstation capacity $\text{CT} \in Z^+$, number $\text{NWS} \in Z^+$ of workstations, difference between largest and smallest idle time $V \in \mathbf{N}$, hazard measure $H \in \mathbf{N}$, demand measure $D \in \mathbf{N}$, and direction change measure $R \in \mathbf{N}$.
>
> *Question*: Is there a partition of P into disjoint sets $P_A, P_B, \ldots, P_{\text{NWS}}$, such that the sum of the sizes of the tasks in each P_X is CT or less, the difference between largest and smallest idle times is V or less, the sum of the hazardous part binary values multiplied by their sequence position is H or less, the sum of the demanded part values multiplied by their sequence position is D or less, the sum of the number of part removal direction changes is R or less, and it obeys the precedence constraints?
>
> ***Proof*:** DLBP $\in$ NP. Given an instance, it can be verified in polynomial time if the answer is "yes" by counting the number of disjoint sets and showing that they are NWS or less, summing the sizes of the tasks in each disjoint set and showing that they are CT or less, examining each of the disjoint sets and noting the largest and the smallest idle times, then subtracting the smallest from the largest and showing that this value is V or less, summing the hazardous part binary values multiplied by their sequence position and showing this is H or less, summing the demanded part values multiplied by their sequence position and showing that this

is D or less, summing the number of changes in part removal direction and showing that this is R or less, and checking that each task has no predecessors listed after it in the sequence.

Multiprocessor scheduling $\leq_P$ DLBP. Restrict to multiprocessor scheduling by allowing only instances in which $V = \text{CT}$, $<$ is empty, and $h_x = h_y$, $d_x = d_y$, $r_x = r_y$ $\forall x, y \in P$.

Therefore, the decision version of DLBP is NP-complete in the strong sense.

DLBP has also been shown to be NP-hard [36]. It is easy to see that the bin-packing problem is also similar to DLBP. The bin-packing problem is NP-hard in the strong sense (it includes 3-partition as a special case), which indicates that there is little hope in finding even a pseudopolynomial time optimization algorithm for it. To solve an exponential time problem, Papadimitriou and Steiglitz [43] propose one of the following approaches be used:

Approximation. Application of an algorithm that quickly finds a suboptimal solution that is within a certain (known) range or percentage of the optimal solution.

Probabilistic. Application of an algorithm that provably yields good average runtime behavior for a distribution of the problem instances.

Special cases. Application of an algorithm that is provably fast if the problem instances belong to a certain special case.

Exponential. Strictly speaking, pseudopolynomial time algorithms are exponential; also, many effective search techniques (e.g., branch-and-bound, simplex) have exponential worst-case complexity.

Local search. Recognized to be one of the most effective search techniques for hard combinatorial optimization problems, local (or *neighborhood*) search is the discrete analog of "hill climbing."

Heuristic. Application of an algorithm that works reasonably well on many cases, but cannot be proven to always be fast or optimal.

In this chapter, several promising heuristic techniques are demonstrated using some of the few DLBP instances found in the literature at this time.

6.8 Graphical Representation

DLBP can be represented by a directed graph (digraph). The DLBP digraph is *strongly connected* when there are no precedence constraints and *weakly connected* otherwise. A modified digraph-based disassembly diagram is effective for displaying the additional and relevant information in a disassembly problem's precedence diagram [34]. It is based on the familiar format found

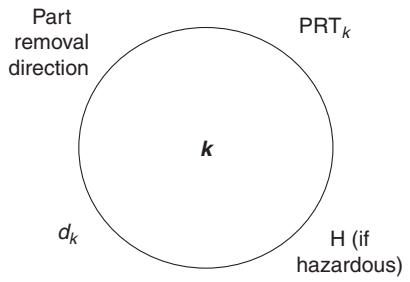

FIGURE 6.1
Proposed disassembly task/part vertex representation.

in assembly line balancing problems (as depicted in Ref. 10; see Ref. 27 for other representations), which consists of a circular symbol (the *vertex*) with the task identification number in the center and the part removal time listed outside and above. This is selectively enhanced to provide greater usability in application to disassembly problems. The part removal time is kept outside of the vertex, but in the upper right corner. Working around in a clockwise manner, the additional considerations found in disassembly are listed using, as a default, the order of priority as given in this chapter; that is, "H" (only if the part is labeled as hazardous, else blank), the part's demand, and the part's removal direction (Figure 6.1).

This representation provides a reference to all of a given problem's
AQ3 disassembly-unique quantitative data and concentrates this data in one location while still enabling all of the graphical relationship information that a traditional precedence diagram portrays. *Directed edges* (*arcs*) continue to be depicted as solid lines terminated with arrows. The arrow's direction points (top to bottom, or left to right) toward subsequent disassembly tasks; that is, the arrows leave the predecessor parts. This format tends to be more angular, similar to an electronics schematic or a flowchart.

Another proposed modification is the use of broken lines to express OR relations by connecting them to each other. Both AND and OR relationships are depicted in Figure 6.2; that is, remove (1 AND 2) OR (2 AND 3) prior to 4 (note that, e.g., part 1 takes 2 s to remove, is hazardous, has a demand of 7, and can only be removed in direction $-y$).

Although all parts should be included in the diagram, only parts with demands (per Section 4.5, it is desirable to remove these parts and their predecessors) or parts that are predecessors to subsequently demanded parts are required to be listed when incomplete disassembly is possible. Parts without demands that do not precede other parts having demands can be deleted from the diagram to conserve space—however, all of any deleted part's additional information (PRT_k, h_k, and r_k) will no longer be available to the reader. Since, in this case, the diagram would be incomplete, the beginning and end of the disassembly precedence diagram can optionally be depicted using flowchart-type terminator boxes, with the start box used to

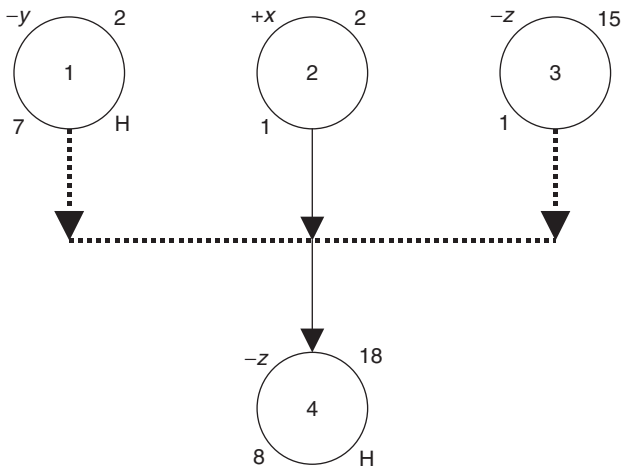

FIGURE 6.2
AND and OR example depicting the requirement to remove (1 AND 2) OR (2 AND 3) prior to 4.

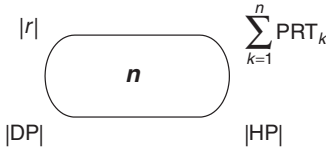

FIGURE 6.3
Disassembly precedence diagram *start* terminator box (optional).

contain summary information about the product being disassembled. The total number of parts is listed in the center and, working around the outside in a clockwise manner, the total part removal time for the entire product is listed in the upper right corner, then the number of hazardous parts, the number of demanded parts, and the number of part removal directions; see Figure 6.3.

Similar to the set of hazardous parts, the set of demanded parts is defined as

$$DP = \{k : d_k \neq 0 \ \forall k \in P\} \tag{6.32}$$

6.9 Case Study Instances

Since DLBP is a recent problem, very few instances exist to study the performance of different heuristic solutions. Four that have been used include: the personal computer problem instance, the 10-part problem instance, the cell phone problem instance, and a variable-size, known-solution benchmark set of instances.

6.9.1 Personal Computer Problem Instance

Gungor and Gupta developed the personal computer problem instance based on an actual study [19]. This practical and relevant example from the literature consists of the data for the disassembly of a personal computer (PC) as shown in Table 6.1 where the objective is to completely disassemble a PC consisting of 8 subassemblies on a paced disassembly line operating at a speed, which allows 40 s for each workstation to perform its required disassembly tasks (see Figure 6.4; note the dotted lines as described in Section 8, which represent the OR relationship in the product; e.g., remove part (2 OR 3) prior to part 6).

6.9.2 10-Part Problem Instance

Kongar and Gupta provided the basis for the 10-part DLBP instance [25]. McGovern and Gupta then modified this instance from its original use in disassembly sequencing and augmented it with disassembly line-specific attributes [35]. Here the objective is to completely disassemble a notional product (see Figure 6.5) consisting of $n = 10$ components and several precedence relationships (e.g., parts 5 AND 6 need to be removed prior to part 7). The problem and its data were modified for use on a paced disassembly line operating at a speed, which allows CT = 40 s for each workstation to perform its required disassembly tasks. This example consists of the data for the disassembly of a product as shown in Table 6.2.

6.9.3 Cell Phone Problem Instance

With the growth of cellular telephone use and technology, rapid changes in usage, technology, and features have prompted the entry of new models on a regular basis. Unwanted cell phones typically end up in landfills and usually contain numerous hazardous parts including mercury, cadmium, lead, gallium arsenide, and beryllium; any of which can pose a threat to the environment. McGovern and Gupta [35, 37] selected a 2001 model

TABLE 6.1

Knowledge Base of the Personal Computer Example

Task	Part Removal Description	Time	Hazardous	Demand	Direction
1	PC top cover	14	No	360	$-x$
2	Floppy drive	10	No	500	$+x$
3	Hard drive	12	No	620	$-x$
4	Back plane	18	No	480	$\pm x, \pm y$
5	PCI cards	23	No	540	$+y$
6	RAM modules (2)	16	No	750	$+z$
7	Power supply	20	Yes	295	$\pm x, \pm y$
8	Motherboard	36	No	720	$+z$

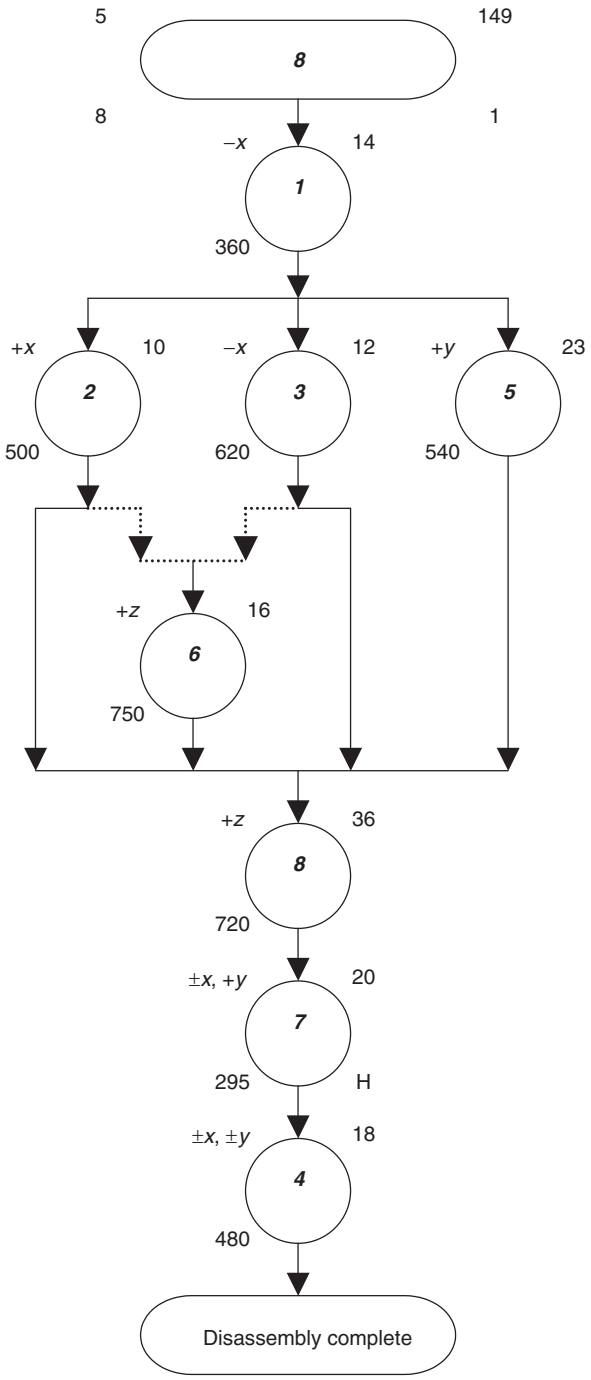

FIGURE 6.4
Personal computer precedence relationships.

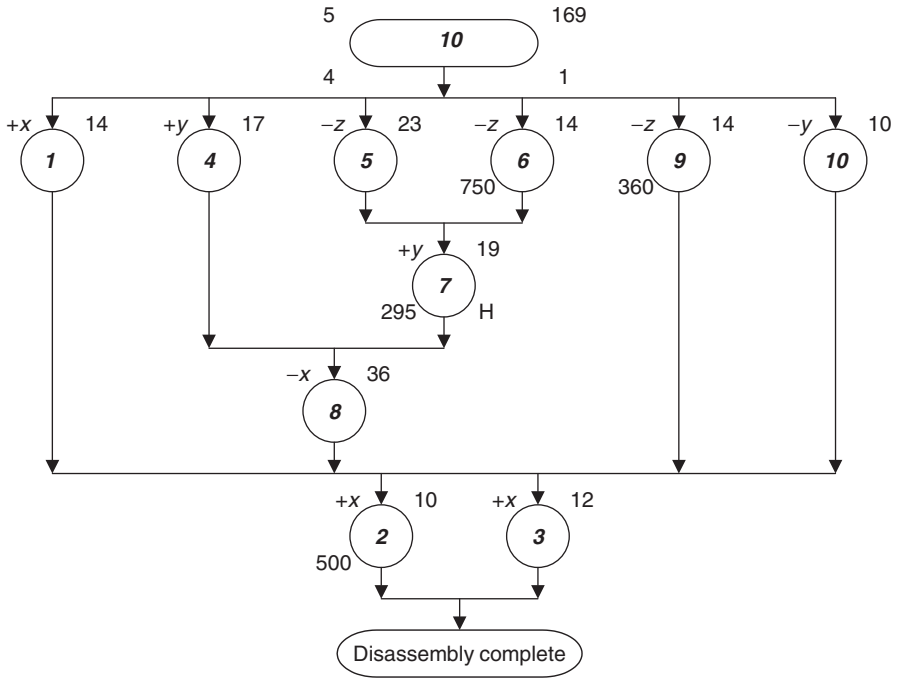

FIGURE 6.5
10-part product precedence relationships.

TABLE 6.2

Knowledge Base of the 10-Part Problem Instance

Task	Time	Hazardous	Demand	Direction
1	14	No	No	$+y$
2	10	No	500	$+x$
3	12	No	No	$+x$
4	17	No	No	$+y$
5	23	No	No	$-z$
6	14	No	750	$-z$
7	19	Yes	295	$+y$
8	36	No	No	$-x$
9	14	No	360	$-z$
10	10	No	No	–y

year Samsung SCH-3500 cell phone for disassembly analysis. The result is an appropriate, real-world instance consisting of $n = 25$ components having several precedence relationships. The data set includes a paced disassembly line operating at a speed that allows CT = 18 s per workstation. Collected data on the SCH-3500 is listed in Table 6.3. Demand is estimated based on part value or recycling value; part removal times and precedence relationships (Figure 6.6) were determined experimentally; part removal times were repeatedly collected until a consistent part removal performance was attained.

TABLE 6.3

Knowledge Base of the Cell Phone Problem Instance

Task	Part Removal Description	Time	Hazardous	Demand	Direction
1	Antenna	3	Yes	4	$+y$
2	Battery	2	Yes	7	$-y$
3	Antenna guide	3	No	1	$-z$
4	Bolt (type 1) a	10	No	1	$-z$
5	Bolt (type 1) b	10	No	1	$-z$
6	Bolt (type 2) 1	15	No	1	$-z$
7	Bolt (type 2) 2	15	No	1	$-z$
8	Bolt (type 2) 3	15	No	1	$-z$
9	Bolt (type 2) 4	15	No	1	$-z$
10	Clip	2	No	2	$+z$
11	Rubber seal	2	No	1	$+z$
12	Speaker	2	Yes	4	$+z$
13	White cable	2	No	1	$-z$
14	Red/blue cable	2	No	1	$+y$
15	Orange cable	2	No	1	$+x$
16	Metal top	2	No	1	$+y$
17	Front cover	2	No	2	$+z$
18	Back cover	3	No	2	$-z$
19	Circuit board	18	Yes	8	$-z$
20	Plastic screen	5	No	1	$+z$
21	Keyboard	1	No	4	$+z$
22	LCD	5	No	6	$+z$
23	Subkeyboard	15	Yes	7	$+z$
24	Internal IC	2	No	1	$+z$
25	Microphone	2	Yes	4	$+z$

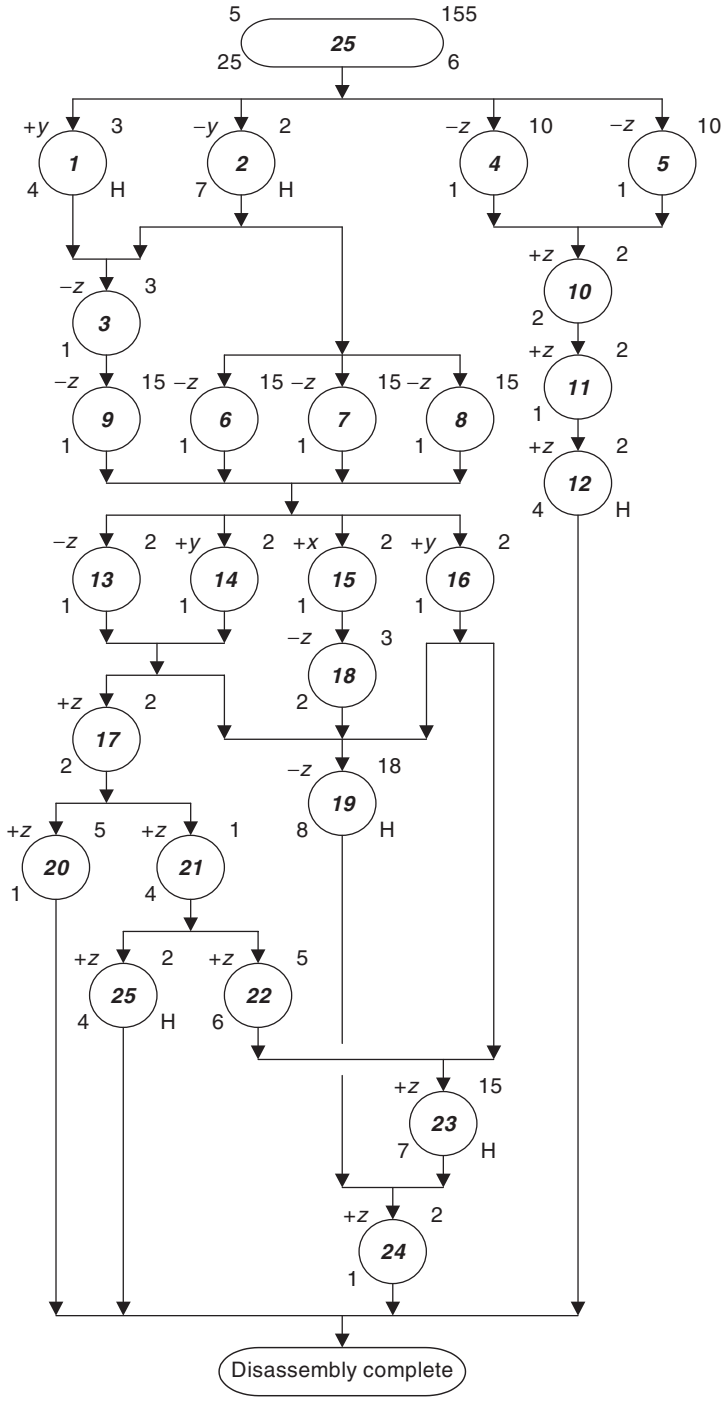

FIGURE 6.6
Cell phone precedence relationships.

6.9.4 DLBP *A Priori* Optimal Solution Benchmark

Any DLBP solution methodology needs to be applied to a collection of test cases to demonstrate its performance as well as its limitations. In addition, new methodologies must first have their developed software thoroughly tested by undergoing a verification and validation (V&V) process. Verification consists of providing a wide range of inputs to a module of the software to ensure proper operation of an individual software component, while validation determines whether or not the program as a whole provides a correct output for given input, necessitating (in the case of DLBP and similar problems) varying-size data sets having known optimal results.

Benchmark data sets are common for many NP-complete problems, such as *Oliver30* and *RY48P* for application to the traveling salesman problem (TSP) and *Nugent15/20/30*, *Elshafei19* and *Krarup30* for the quadratic assignment problem. Unfortunately, because of their size and their design, most of these existing data sets have no known optimal answer and new solutions are not compared to the optimal solution, but rather the best-known solution to date. In addition, since DLBP is a recently defined problem, no appropriate benchmark data sets exist. It was necessary to develop a set of instances for the DLBP to evaluate DLBP heuristics. We propose a known optimal solution benchmark line balance data set to determine efficacy (a method's effectiveness in finding good solutions).

This size-independent *a priori* benchmark data set was generated [35] based on the following. Because, in general, solutions to larger and larger instances cannot be verified as optimal (due to the time complexity of exhaustive search), it is proposed that instances be generated in such a way to always provide a known solution. This was done by using part times consisting exclusively of prime numbers further selected to ensure that no combinations of these part removal times allowed for any equal summations (to reduce the number of possible optimal solutions). For example, part removal times 1, 3, 5, and 7 and CT = 16 would have minimum idle time solutions of not only one 1, one 3, one 5, and one 7 at each workstation, but various additional combinations of these as well since 1 + 7 = 3 + 5 = ½ CT. Subsequently, the chosen instances were made up of parts with removal times of 3, 5, 7, and 11 and CT = 26. As a result, the optimal balance for all subsequent instances would consist of a perfect balance of precedence-preserving combinations of 3, 5, 7, and 11 at each workstation with idle times of zero. This *a priori* data set is then constrained by

$$n = x \cdot |PRT : x \in Z^{+} \tag{6.33}$$

To further complicate the data (i.e., provide a large, feasible search space), only one part was listed as hazardous and this was one of the parts with the largest part removal time (the last one listed in the original data). In addition, one part (the last listed, second-largest part removal time component) was listed as being demanded. This was done so that only the hazardous and the

demand sequencing would be demonstrated, while providing a slight solution sequence disadvantage to any purely greedy methodology (since two parts with part removal times of 3 and 5 are needed along with the larger part removal time parts to reach F^*, assigning hazardous and demanded parts to those smaller part removal time parts may allow some methodologies to artificially obtain the initial F^* single workstation sequence). From each part removal time size, the first listed part was selected to have a removal direction differing from the other parts with the same part removal time. This was done to demonstrate direction selection while requiring any solution-generating methodology to move these first parts of each part removal time size encountered to the end of the sequence (i.e., into the last workstation) to obtain the optimal direction value of $R^* = 1$ (i.e., if the solution technique being evaluated is able to successfully place the hazardous and demanded parts toward the front of the sequence). There were also no precedence constraints placed on the sequence, a deletion that further challenges any method's ability to attain an optimal solution. Known optimal results include $F^* = 0$, $H^* = 1$, $D^* = 2$, $R^* = 1$. While this chapter makes use of data with $|\text{PRT}| = 4$ unique part removal times, in general, for any n parts consisting of this type of data, the following can be calculated

$$\text{NWS}^* = \frac{n}{|\text{PRT}|} \tag{6.34}$$

$$\text{NWS}_{\text{nom}} = n \tag{6.35}$$

$$I^* = 0 \tag{6.36}$$

$$I_{\text{nom}} = \frac{n \cdot \text{CT} \cdot (|\text{PRT}| - 1)}{|\text{PRT}|} \tag{6.37}$$

$$F^* = 0 \tag{6.38}$$

with F_{nom} given by Formula 6.11. Hazard measures and values are given by

$$h_k = \begin{cases} 1, & k = n \\ 0, & \text{otherwise} \end{cases} \tag{6.39}$$

$$H^* = 1 \tag{6.40}$$

$$H_{\text{nom}} = n \tag{6.41}$$

with demand measures and values given by

$$d_k = \begin{cases} 1, & k = \dfrac{n \cdot (|\text{PRT}| - 1)}{|\text{PRT}|} \\ 0, & \text{otherwise} \end{cases} \tag{6.42}$$

$$D^* = \begin{cases} 2, & H = 1 \\ 1, & \text{otherwise} \end{cases} \tag{6.43}$$

$$D_{\text{nom}} = \begin{cases} n-1, & H = n \\ n, & \text{otherwise} \end{cases} \tag{6.44}$$

and part removal direction measures and values given by

$$r_k = \begin{cases} 1, & k = 1, \dfrac{n}{|\text{PRT}|} + 1, \dfrac{2n}{|\text{PRT}|} + 1, \ldots, \dfrac{(|\text{PRT}| - 1) \cdot n}{|\text{PRT}|} + 1 \\ 0, & \text{otherwise} \end{cases} \tag{6.45}$$

$$R^* = 1 \tag{6.46}$$

$$R_{\text{nom}} = \begin{cases} 0, & n = |\text{PRT}| \\ 2 \cdot |\text{PRT}| - 1, & n = 2 \cdot |\text{PRT}| \\ 2 \cdot |\text{PRT}|, & \text{otherwise} \end{cases} \tag{6.47}$$

Because $|\text{PRT}| = 4$ in this chapter, each part removal time is generated by

$$\text{PRT}_k = \begin{cases} 3, 0 < k \le \dfrac{n}{4} \\ 5, \dfrac{n}{4} < k \le \dfrac{n}{2} \\ 7, \dfrac{n}{2} < k \le \dfrac{3n}{4} \\ 11, \dfrac{3n}{4} < k \le n \end{cases} \tag{6.48}$$

Although the demand values as generated by Formula 6.42 are the preferred representation (due to the fact that the resulting small numerical values make it easy to interpret demand efficacy since $D = k$), algorithms that allow incomplete disassembly may terminate after placing the single demanded

part in the solution sequence. In this case, Formulae 6.42 through 6.44 may be modified to give

$$d_k = \begin{cases} 2, & k = \dfrac{n \cdot (|\text{PRT}| - 1)}{|\text{PRT}|} \\ 1, & \text{otherwise} \end{cases} \tag{6.49}$$

$$D^* = \begin{cases} 2 + \sum_{p=1}^{n} p, & H{=}1 \\ 1 + \sum_{p=1}^{n} p, & \text{otherwise} \end{cases} \tag{6.50}$$

$$D_{\text{nom}} = \begin{cases} n - 1 + \sum_{p=1}^{n} p, & H = n \\ n + \sum_{p=1}^{n} p, & \text{otherwise} \end{cases} \tag{6.51}$$

Note that a data set containing parts with equal PRTs and no precedence constraints will have more than one optimal solution. To properly gauge the performance of any solution-generating technique on the DLBP *a priori* data, the size of the optimal solution set needs to be quantified. From probability theory we know that, for example, with $n = 12$ and $|\text{PRT}| = 4$, the size of the set of optimally balanced solutions $|F^*|$ when using the DLBP *a priori* data could be calculated as $(12 \cdot 9 \cdot 6 \cdot 3) \cdot (8 \cdot 6 \cdot 4 \cdot 2) \cdot (4 \cdot 3 \cdot 2 \cdot 1) = 17{,}915{,}904$ from Table 6.4.

Grouping these counts by workstation and reversing their ordering enables one to recognize a pattern more easily.

$$\begin{array}{c} 1\ \ 2\ \ 3\ \ 4) \\ (2\ \ 4\ \ 6\ \ 8) \\ (3\ \ 6\ \ 9\ \ 12) \end{array}$$

It can be seen that the first row can be generalized as $(1 \cdot 2 \cdot 3 \cdot \cdots \cdot |\text{PRT}|)$, the second as $(2 \cdot 4 \cdot 6 \cdot \cdots \cdot 2 \cdot |\text{PRT}|)$, and the third as $(3 \cdot 6 \cdot 9 \cdot \cdots \cdot 3 \cdot |\text{PRT}|)$. Expanding in this way, the number of optimally balanced solutions can be written as

$$|F^*| = (1 \cdot 2 \cdot 3 \cdot \cdots \cdot (1 \cdot |\text{PRT}|)) \cdot (2 \cdot 4 \cdot 6 \cdot \cdots \cdot (2|\text{PRT}|)) \cdot \cdots \cdot \left(\frac{n}{|\text{PRT}|} \cdot \frac{2n}{|\text{PRT}|} \cdot \frac{3n}{|\text{PRT}|} \cdots \cdot n \right)$$

TABLE 6.4

Number of Possible Entries in Each Element Position Resulting in Perfect Balance Using the DLBP *A Priori* Data with $n = 12$ and $|\text{PRT}| = 4$

k	1	2	3	4	5	6	7	8	9	10	11	12
Count	12	9	6	3	8	6	4	2	4	3	2	1

This can be written as

$$|F^*| = \prod_{x=1}^{|\mathrm{PRT}|} x \cdot \prod_{x=1}^{|\mathrm{PRT}|} 2x \cdot \prod_{x=1}^{|\mathrm{PRT}|} 3x \cdot \cdots \cdot \prod_{x=1}^{|\mathrm{PRT}|} \frac{n}{|\mathrm{PRT}|} \cdot x$$

and finally as

$$|F^*| = \prod_{x=1}^{|\mathrm{PRT}|} x^{\frac{n}{|\mathrm{PRT}|}} \cdot \prod_{y=1}^{\frac{n}{|\mathrm{PRT}|}} y$$

or

$$|F^*| = \prod_{x=1}^{|\mathrm{PRT}|} \prod_{y=1}^{\frac{n}{|\mathrm{PRT}|}} x^{\frac{n}{|\mathrm{PRT}|}} \cdot y \tag{6.52}$$

Since $|\mathrm{PRT}| = 4$ in this chapter, Formula 6.52 becomes

$$|F^*| = \prod_{x=1}^{4} \prod_{y=1}^{\frac{n}{4}} x^{\frac{n}{4}} \cdot y \tag{6.53}$$

In our example with $n = 12$ and $|\mathrm{PRT}| = 4$, Formula 6.53 is solved as

$$|F^*| = \prod_{x=1}^{4} \prod_{y=1}^{\frac{12}{4}} x^{\frac{12}{4}} \cdot y = \prod_{x=1}^{4} \prod_{y=1}^{3} x^3 \cdot y = \prod_{x=1}^{4} x^3 \cdot (1 \cdot 2 \cdot 3)$$

or

$$|F^*| = 1 \cdot 1 \cdot 1 \cdot (1 \cdot 2 \cdot 3) \cdot 2 \cdot 2 \cdot 2 \cdot (1 \cdot 2 \cdot 3) \cdot 3 \cdot 3 \cdot 3 \cdot (1 \cdot 2 \cdot 3) \cdot 4 \cdot 4 \cdot 4 \cdot (1 \cdot 2 \cdot 3)$$

which, when rearranged, can be written as the more familiar

$$|F^*| = (12 \cdot 9 \cdot 6 \cdot 3) \cdot (8 \cdot 6 \cdot 4 \cdot 2) \cdot (4 \cdot 3 \cdot 2 \cdot 1) = 17{,}915{,}904$$

Even when all objectives are considered, there still exist multiple optimal solutions, again due to the use of a data set containing parts with equal PRTs and no precedence constraints. Using probability theory with the example having $n = 12$ and $|\mathrm{PRT}| = 4$, it is known that the size of the set of solutions optimal in *F, H, D,* and *R*, $|F^* \cap H^* \cap D^* \cap R^*|$, when using the DLBP *a priori* data can be calculated as $(1 \cdot 1 \cdot 6 \cdot 3) \cdot (4 \cdot 3 \cdot 2 \cdot 1) \cdot (4 \cdot 3 \cdot 2 \cdot 1) = 10{,}368$ from Table 6.5.

TABLE 6.5

Number of possible entries in each element position resulting in optimal in *F, H, D,* and *R* using *a priori* data with $n = 12$ and $|\mathrm{PRT}| = 4$

k	1	2	3	4	5	6	7	8	9	10	11	12
Count	1	1	6	3	4	3	2	1	4	3	2	1

Repeating the technique of grouping these counts by workstation and reversing their ordering again reveals a pattern.

$$(1\ \ 2\ \ 3\ \ 4)$$
$$(1\ \ 2\ \ 3\ \ 4)$$
$$(3\ \ 6\ \ 1\ \ 1)$$

The middle elements will always be the same as those given by Formula 6.52, but with two fewer sets (due to different first and last workstation elements). The first row is always $(1 \cdot 2 \cdot 3 \cdot \cdots \cdot |\text{PRT}|)$ since the directional elements in the *a priori* data should always be together at the end (the beginning in this case since we reversed the sequence for readability) of any optimal solution sequence. The last row (again, reversed) is always $((n/|\text{PRT}|) \cdot (2n/|\text{PRT}|) \cdot (3n/|\text{PRT}|) \cdot \cdots \cdot ((|\text{PRT}|-2) \cdot n/|\text{PRT}|) \cdot 1 \cdot 1)$ since there is only one hazardous part (optimal element position $k = 1$) and only one demanded part (optimal element position $k = 2$). Combining these components, the number of fully optimal solutions can be written as

$$|F^* \cap H^* \cap D^* \cap R^*| = (1 \cdot 2 \cdot 3 \cdot \cdots \cdot |\text{PRT}|) \cdot$$

$$(1 \cdot 2 \cdot 3 \cdot \cdots \cdot (1 \cdot |\text{PRT}|)) \cdot (2 \cdot 4 \cdot 6 \cdot \cdots \cdot (2 \cdot |\text{PRT}|)) \cdot \cdots \cdot 1 \cdot \left(\frac{n}{|\text{PRT}|} - 2\right) \cdot 2$$
$$\cdot \left(\frac{n}{|\text{PRT}|} - 2\right) \cdot 3 \cdot \left(\frac{n}{|\text{PRT}|} - 2\right) \cdot \ldots \cdot |\text{PRT}| \cdot \left(\frac{n}{|\text{PRT}|} - 2\right)\Bigg)\Bigg)$$
$$\cdot \left(\left(\frac{n}{|\text{PRT}|}\right) \cdot \left(\frac{2n}{|\text{PRT}|}\right) \cdot \left(\frac{3n}{|\text{PRT}|}\right) \cdot \ldots \cdot \left(\frac{(|\text{PRT}|-2) \cdot n}{|\text{PRT}|}\right) \cdot 1 \cdot 1\right)$$

By replacing the second term with a modified version of Formula 6.52 and simplifying the first and third terms, this can be written as

$$|F^* \cap H^* \cap D^* \cap R^*| = \left(\prod_{y=1}^{|\text{PRT}|} y\right) \cdot \prod_{y=1}^{|\text{PRT}|} \prod_{z=1}^{\frac{n}{|\text{PRT}|}-2} y^{\frac{n}{|\text{PRT}|}-2} \cdot z \cdot \left(\prod_{x=1}^{|\text{PRT}|-2} \frac{n}{|\text{PRT}|} \cdot x\right)$$

Expanding the second term gives

$$|F^* \cap H^* \cap D^* \cap R^*| = \left(\prod_{y=1}^{|\text{PRT}|} y\right) \cdot \left(\prod_{y=1}^{|\text{PRT}|} y^{\frac{n}{|\text{PRT}|}-2} \cdot \prod_{z=1}^{\frac{n}{|\text{PRT}|}-2} z\right) \left(\prod_{x=1}^{|\text{PRT}|-2} \frac{n}{|\text{PRT}|} \cdot x\right)$$

Combining the first and second terms results in

$$|F^* \cap H^* \cap D^* \cap R^*| = \left(\prod_{y=1}^{|\text{PRT}|} y^{\frac{n}{|\text{PRT}|}-1} \left(\prod_{z=1}^{\frac{n}{|\text{PRT}|}-2} z\right)\right) \cdot \left(\prod_{x=1}^{|\text{PRT}|-2} \frac{n}{|\text{PRT}|} \cdot x\right)$$

or

$$|F^* \cap H^* \cap D^* \cap R^*| = \prod_{x=1}^{|\mathrm{PRT}|-2} \frac{nx}{|\mathrm{PRT}|} \cdot \prod_{y=1}^{|\mathrm{PRT}|} \prod_{z=1}^{\frac{n}{|\mathrm{PRT}|}-2} y^{\frac{n}{|\mathrm{PRT}|}-1} \cdot z$$

with a constraint that

$$\frac{n}{|\mathrm{PRT}|} > 2$$

Alternatively, this can be written as

$$|F^* \cap H^* \cap D^* \cap R^*| = \prod_{x=1}^{|\mathrm{PRT}|-2} \frac{nx}{|\mathrm{PRT}|} \cdot \prod_{y=1}^{|\mathrm{PRT}|} \prod_{z=1}^{a} y^{\frac{n}{|\mathrm{PRT}|}-1} \cdot z \qquad (6.54)$$

where

$$a = \begin{cases} \frac{n}{|\mathrm{PRT}|} - 2, & \text{if } \frac{n}{|\mathrm{PRT}|} > 2 \\ 1, & \text{otherwise} \end{cases}$$

Since $|\mathrm{PRT}| = 4$ in this chapter, Formula 6.54 becomes

$$|F^* \cap H^* \cap D^* \cap R^*| = \prod_{x=1}^{2} \frac{nx}{4} \cdot \prod_{y=1}^{4} \prod_{z=1}^{\frac{n}{4}-2} y^{\frac{n}{4}-1} \cdot z \qquad (6.55)$$

In our example with $n = 12$ and $|\mathrm{PRT}| = 4$, Formula 6.55 is solved as

$$|F^* \cap H^* \cap D^* \cap R^*| = \prod_{x=1}^{2} 3x \cdot \prod_{y=1}^{4} \prod_{z=1}^{1} y^2 \cdot z$$

or

$$|F^* \cap H^* \cap D^* \cap R^*| = \prod_{x=1}^{2} 3x \cdot \prod_{y=1}^{4} y^2$$

giving

$$|F^* \cap H^* \cap D^* \cap R^*| = ((3 \cdot 1) \cdot (3 \cdot 2)) \cdot ((1 \cdot 1) \cdot (2 \cdot 2) \cdot (3 \cdot 3) \cdot (4 \cdot 4))$$

which, when rearranged, can be written as the more familiar

$$|F^* \cap H^* \cap D^* \cap R^*| = (1 \cdot 1 \cdot 6 \cdot 3) \cdot (4 \cdot 3 \cdot 2 \cdot 1) \cdot (4 \cdot 3 \cdot 2 \cdot 1) = 10{,}368$$

Although the sizes of both DLBP *a priori* optimal solution sets are quite large in these examples, they are also significantly smaller than the search space

TABLE 6.6

Comparison of Possible Solutions to Optimal Solutions for a Given n Using the DLBP *A Priori* Data

n	$n!$	Number Optimal in Balance	Number Optimal in All	Percentage Optimal in Balance (%)	Percentage Optimal in All (%)
4	24	24	2	100.00	8.33
8	40,320	9,216	48	22.86	0.12
12	479,001,600	17,915,904	10,368	3.74	0.00
16	2.09228E+13	1.10075E+11	7,077,888	0.53	0.00

of $n! = 479{,}001{,}600$. As shown in Table 6.6, the number of solutions that are optimal in balance alone goes from 100% of n at $n = 4$, to 22.9% at $n = 8$, and to less than 1% at $n = 16$; as n grows, this percentage gets closer and closer to 0%. The number of solutions optimal in all objectives goes from less than 8.3% of n at $n = 4$, to 0.12% at $n = 8$, dropping to effectively 0% at $n = 16$; again, as n grows, the percentage of optimal solutions gets closer and closer to zero.

6.10 The Combinatorial Optimization Searches

Five solution techniques (exhaustive search, GA, ant colony optimization, greedy/adjacent element hill-climbing hybrid heuristic, and the H–K general-purpose heuristic) are considered, then the four heuristics are selected for analysis and evaluation using one of the instances from Section 6.9 (although the DLBP *a priori* benchmark data are also used with exhaustive search, this is primarily to demonstrate exponential growth rather than to allow for a thorough analysis of the chosen algorithm). Sections 6.10.5.4 and 6.10.5.5 demonstrate the graphical analysis tools first developed by McGovern and Gupta [38].

For consistency in software architecture and data structures, the authors wrote all of the search algorithms; no off-the-shelf software was used. In addition, any sections of software code that could be used by multiple programs were reused. All computer programs were written in C++ and run on a 1.6 GHz PM × 86-family workstation. After engineering, each program was first investigated on a variety of test cases for V&V purposes. All of the combinatorial optimization computer software was run at least three times to obtain an average of the computation time. Solution data for use in efficacy analysis were collected a minimum of five times when methodologies

possessed a probabilistic component (ACO and GA) with the results averaged to avoid reporting unusually favorable or unusually poor results.

6.10.1 Exhaustive Search

6.10.1.1 Exhaustive Search Model Description

An exhaustive search algorithm was developed to confirm the exponential time growth of exhaustive search, to provide a time complexity benchmark for the subsequent algorithm studies, and to determine the optimal solutions for any data set up to a given size (based on a reasonable search time). The exhaustive algorithm was built around a recursive function (Figure 6.7) that generated all permutations of a sequence of n numbers (given the input $q = n$ when it is first called) with each number representing a disassembly task and the order of the numbers representing the disassembly sequence. The purpose of this exhaustive algorithm is to find every subset of the data set (every permutation) and from all those subsets, find the solution set that best satisfies the performance requirements by checking against all other permutations as described in Section 6.5.

```
Procedure GENERATE (q){
   IF ((q = = 0) ∧ (PRECEDENCE_PASS)){

      CALC_PERFORMANCE

      IF (FIRST_PERMUTATION_GENERATED){
         SET best_solution := new_solution
       }

      IF (BETTER_SOLUTION (new_solution, best_solution)){
         SET best_solution := new_solution
       }
   }

   ∀ k∈ P{
       EXCHANGE (k, q – 1)
       GENERATE (q – 1)
       EXCHANGE (k, q – 1)
   }
   RETURN
}

Procedure EXCHANGE (x, y){
      SET temp := PSx
      SET PSx := PSy
      SET PSy := temp
      RETURN
}
```

FIGURE 6.7
Recursive backtracking algorithm to generate all permutations of n numbers.

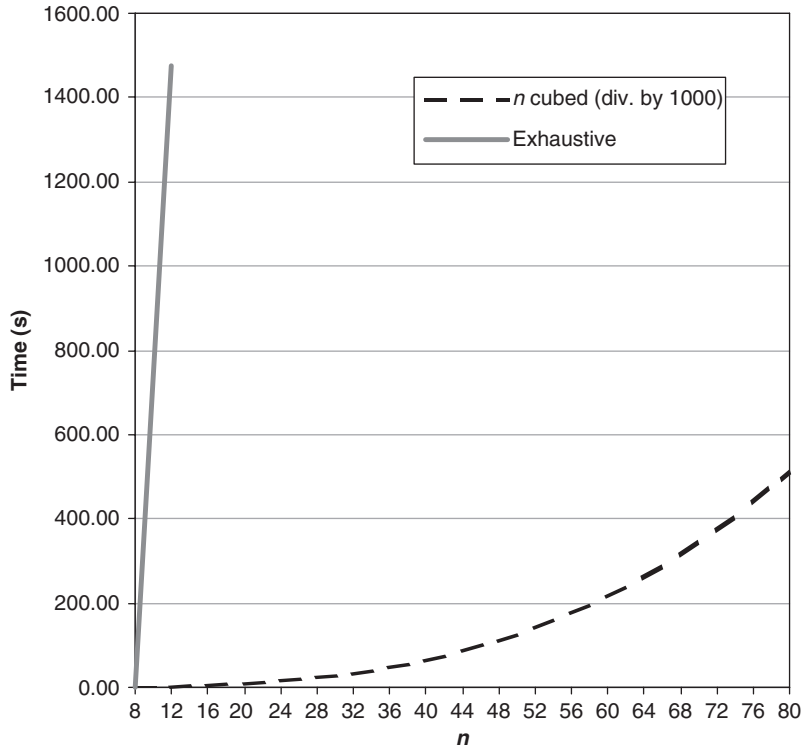

FIGURE 6.8
DLBP exhaustive search time complexity.

6.10.1.2 Exhaustive Search Algorithm Numerical Results

Figure 6.8 provides the plot of runtime versus instance size using the DLBP *a priori* benchmark for the exhaustive algorithm and an n^3 curve for growth comparison and to demonstrate how the exhaustive search runtime rapidly increases with instance size. Using the DLBP *a priori* instances, $n = 8$ averaged 0.10 s, while $n = 12$ averaged 1476.44 s (just under 25 min). At this rate it could be conservatively estimated (assuming a 10-fold increase for each single increment of n; this approximates the observed growth but is actually slower than its true $n!$ growth) that $n = 13$ would take over 4 h, $n = 14$ would take almost 2 days, $n = 15$ would run over 2 weeks, $n = 16$ almost 6 months, $n = 17$ over 4.5 years, and so forth. Faster computers and programming tricks can speed this up but the growth in exhaustive search is exponential nonetheless. Using the DLBP *a priori* instances in the range of $8 \leq n \leq 80$, only $n \leq 12$ sized instances were found to be solved in a reasonable amount of time using the recursive exhaustive search algorithm. The algorithm's time complexity was seen to increase with instance size; in fact, it was seen to be in proportion to $1.199(n!)$ (using the actual growth rate gives an $n = 16$ runtime of 2.04 years).

6.10.2 Genetic Algorithm

6.10.2.1 The DLBP Genetic Algorithm and Model Description

A GA (a parallel neighborhood, stochastic-directed search technique) provides an environment where solutions continuously crossbreed, mutate, and compete with one another until they evolve into an optimal or near-optimal solution. Owing to its structure and search method, a GA is often able to find a global solution, unlike many other heuristics that use hill climbing to find a best solution nearby, resulting only in a local optima. In addition, a GA does not need specific details about a problem nor is the problem's structure relevant; a function can be linear, nonlinear, stochastic, combinatorial, noisy, etc.

GA has a solution structure, defined as a *chromosome*, which is made up of *genes* and generated by two *parent* chromosomes from the *pool* of solutions, each having its own measure of *fitness*. New solutions are generated from old using the techniques of *crossover* (sever parents genes and swap severed sections) and *mutation* (randomly vary genes within a chromosome). Typically, the main challenge with any GA implementation is to determine a chromosome representation that remains valid after each generation.

For DLBP, the chromosome (solution) consisted of a sequence of genes (parts). A *population*, or pool, of size N was used. Only feasible disassembly sequences were considered as members of the population or as offspring. The fitness (F) and NWS were computed for each chromosome using the same method for solution performance determination as in the exhaustive search (H, D, and R calculations are not listed in the interest of space; see Section 6.5 for multicriteria treatment and Section 6.6 for the actual equations).

6.10.2.2 DLBP-Specific Genetic Algorithm Architecture

The GA for DLBP was constructed as follows [39] [32]. An initial, feasible population was randomly generated and the fitness of each chromosome in this generation was calculated. An even integer of $R_x \cdot N$ parents was randomly selected for crossover to produce $R_x \cdot N$ offspring (offspring make up $R_x \cdot N \cdot 100\%$ percent of each generation's population). An elegant crossover, the precedence preservative crossover (PPX) developed by Bierwirth et al. [3], was used to create the offspring. As shown in Figure 6.9, PPX first creates a mask (one for each child, every generation). The mask consists of random 1s and 2s, indicating which parent part information should be taken from. If, for example, the mask for child 1 reads 221211, the first two parts (i.e., from left to right) in parent 2 would make up the first two genes of child 1 (and these parts would be stricken from the parts available to take from both parent 1 and 2); the first available (i.e., not stricken) part in parent 1 would make up gene three of child 1; the next available part in parent 2 would make up gene four of child 1; the last two parts in parent 1 would make up genes five and six of child 1. This technique is repeated using a new mask for child 2. After crossover, mutation is randomly conducted. Mutation was occasionally (based on the R_m value) performed by randomly selecting a single child then

Parent 1: 1 3 2 6 5 8 7 4
Parent 2: 1 2 3 5 6 8 7 4
Mask: 2 2 1 2 1 1 1 2
Child:

Parent 1: 1 3 2 6 5 8 7 4
Parent 2: **1 2** 3 5 6 8 7 4
Mask: 2 2 1 2 1 1 1 2
Child: 1 2

Parent 1: x **3** x 6 5 8 7 4
Parent 2: x x 3 5 6 8 7 4
Mask: 1 2 1 1 1 2
Child: 1 2 3

Parent 1: x x x 6 5 8 7 4
Parent 2: x x x **5** 6 8 7 4
Mask: 2 1 1 1 2
Child: 1 2 3 5

Parent 1: x x x **6** x **8 7** 4
Parent 2: x x x x 6 8 7 4
Mask: 1 1 1 2
Child: 1 2 3 5 6 8 7

Parent 1: x x x x x x x 4
Parent 2: x x x x x x x **4**
Mask: 2
Child: 1 2 3 5 6 8 7 4

FIGURE 6.9
PPX example.

exchanging two of its disassembly tasks while ensuring that precedence is preserved. The $R_x \cdot N$ least fit parents are removed by sorting the entire parent population from worst to best based on fitness. Since the GA saves the best parents from generation to generation and it is possible for duplicates of a solution to be formed using PPX, the solution set could contain multiple copies of the same answer resulting in the algorithm potentially becoming trapped in a local optima. This becomes more likely in a GA with solution constraints (such as precedence requirements) and small populations, both of which are seen in the study in this chapter. To avoid this, DLBP GA was modified to treat duplicate solutions as if they had the worst fitness performance (highest numerical value), relegating them to replacement in the next generation. With this new ordering, the best unique $(1 - R_x) \cdot N$ parents were kept along with all of the $R_x \cdot N$ offspring to make up the next generation, then the process was repeated. To again avoid becoming trapped in local optima, DLBP GA—as with many combinatorial optimization techniques—was run not until a desired level of performance was reached but rather for as many generations as deemed acceptable by the user. Since DLBP GA always keeps the best solutions far from generation to generation, there is no risk of solution drift or bias, and the possibility of mutation allows for a diverse range of possible solution space visits over time.

6.10.2.3 DLBP-Specific Genetic Algorithm Qualitative Modifications

DLBP GA was modified from a general GA in several ways. Instead of the worst portion of the population being selected for crossover, in the DLBP GA all of the population was (randomly) considered for crossover. This better enables the selection of nearby solutions (i.e., solutions similar to the best solutions to date) common in many scheduling problems. Also, mutation was performed only on the children, not the worst parents. This was done to address the small population used in DLBP GA and to counter PPX's tendency to duplicate parents. Finally, duplicate children are sorted to make their deletion from the population likely since there is a tendency for the creation of duplicate solutions (due to PPX) and due to the small population saved from generation to generation.

6.10.2.4 DLBP-Specific Genetic Algorithm Quantitative Modifications

A small population was used (20 versus the more typical 10,000 to 100,000) to minimize data storage requirements and simplify analysis (except for the case study where the generations were varied) while a large number of generations were used (10,000 versus the more typical 10–1000) to compensate for this small population while not being so large as to take an excessive amount of processing time. This was also done to avoid solving all cases to optimality since it was desirable to determine the point at which the DLBP GA performance begins to break down and how that breakdown manifests itself. Lower than the recommended 90% [26], a 60% crossover was selected based on test and analysis. Sixty percent crossover provided better solutions and did so with one-third less processing time. Previous assembly line balancing literature that indicated best results have typically been found with crossover rates of from 0.5 to 0.7 also substantiated the selection of this lower crossover rate. A mutation was performed about 1% of the time. Although some texts recommend 0.01% mutation while applications in journal papers have used as much as 100% mutation, it was found that 1.0% gave excellent algorithm performance for the disassembly line balancing problem.

6.10.2.5 DLBP GA Numerical Results Using the Personal Computer Case Study Instance

DLBP GA was used to solve the PC instance. Since one purpose of this exercise was to demonstrate DLBP GA performance with changes in generation size, the data was run varying the generations and without the use of hazard, demand, or direction data to avoid potential complications in the analysis. Owing to GA's stochastic nature also, it was run a minimum of five times with the results averaged to avoid reporting unusually favorable or unusually poor results.

The GA converged to moderately good solutions very quickly. It was able to find at least one of the four solutions optimal in *F* (Table 6.7) after no more than 10 generations. Hundred generations consistently provided three to four of the *F*-optimal solutions, while 1000 generations almost always resulted in the generation of all four optimal solutions. Similarly, the speed for the C++ implemented program searching 1000 generations of 20 chromosomes; each was just over 1 s on the previously described work station.

The balancing aspect of Formula 6.4 worked extremely well with four equivalent (and, in fact, *F*-optimal) solutions being found. All four, best solutions consisted of a total of four workstations with 2–4 s per workstation of idle time. Therefore, the obtained solutions were optimal in number of workstations and also provided idle times at each workstation of at least 5% but not more than 10% of the total disassembly time of 40 s allocated to each workstation. Note that the search space is 8! or 40,320.

TABLE 6.7

The Four Disassembly Sequence Solutions Optimal in *F*

a)

Part removal sequence	Workstation 1	2	3	4	Time to remove part (in seconds)
1	14				
5	23				
3		12			
6		16			
2		10			
8			36		
7				20	
4				18	
Total time	37	38	36	38	
Idle time	3	2	4	2	

b)

Part removal sequence	Workstation 1	2	3	4	Time to remove part (in seconds)
1	14				
5	23				
3		12			
2		10			
6		16			
8			36		
7				20	
4				18	
Total time	37	38	36	38	
Idle time	3	2	4	2	

c)

Part removal sequence	Workstation 1	2	3	4	Time to remove part (in seconds)
1	14				
5	23				
2		10			
6		16			
3		12			
8			36		
7				20	
4				18	
Total time	37	38	36	38	
Idle time	3	2	4	2	

d)

Part removal sequence	Workstation 1	2	3	4	Time to remove part (in seconds)
1	14				
5	23				
2		10			
3		12			
6		16			
8			36		
7				20	
4				18	
Total time	37	38	36	38	
Idle time	3	2	4	2	

6.10.3 The Ant Colony Optimization Metaheuristic

6.10.3.1 Ant Colony Optimization Model Description

ACO is a probabilistic evolutionary algorithm based on a *distributed autocatalytic* (i.e., positive feedback [9]) *process* (i.e., a process that reinforces itself in a way that causes very rapid convergence [9]) that makes use of *agents* called ants (due to these agents' similar attributes to insects). Just as a colony of ants can find the shortest distance to a food source, these ACO agents work cooperatively toward an optimal problem solution. Multiple agents are placed at multiple starting nodes, such as cities for the traveling salesman problem (or parts for the DLBP). Each of the m ants is allowed to visit all remaining (unvisited) edges as indicated by a Tabu-type list. Each ant's possible subsequent steps (i.e., from a node p to node q giving edge pq, vertices and

arcs in DLBP) are evaluated for desirability and each is assigned a proportionate probability as shown in Formula 6.56 (shown modified for DLBP as described in the next section). Based on these probabilities, the next step in the tour is randomly selected for each ant. After completing an entire tour, all feasible ants are given the equivalent of additional pheromone (in proportion to tour desirability), which is added to each step it has taken on its tour. All paths are then decreased in their pheromone strength according to a measure of evaporation (equal to $1 - \rho$). This process is repeated for $\mathrm{NC}_{\max}$ or until stagnation behavior (where all ants make the same tour) is demonstrated. The ant system/ant-cycle model has been claimed to run in $O(\mathrm{NC} \cdot n^3)$ [9].

6.10.3.2 DLBP-Specific Qualitative Modifications and the DLBP ACO algorithm

The DLBP ACO is a modified ant-cycle algorithm [35]. It is designed around the DLBP problem by accounting for feasibility constraints and addressing multiple objectives. In DLBP ACO, each part is a vertex on the tour with the number of ants being set equal to the number of parts and having one ant uniquely on each part as the starting position. Each ant is allowed to visit all parts not already in the solution. (In the sequence example solution from Section 6.5—the n-tuple $\langle 5, 2, 8, 1, 4, 7, 6, 3\rangle$—at time $t = 3$, component p would represent component 8 while component $q \in \{1, 4, 7, 6, 3\}$ and possible partial solution sequences $\langle 5, 2, 8, 1\rangle$, $\langle 5, 2, 8, 4\rangle$, $\langle 5, 2, 8, 7\rangle$, $\langle 5, 2, 8, 6\rangle$, and $\langle 5, 2, 8, 3\rangle$ would be evaluated for feasibility and balance.) Each ant's possible subsequent steps are evaluated for feasibility and the measure of balance then assigned a proportionate probability as given by Formula 6.56. Infeasible steps receive a probability of zero and any ants having only infeasible subsequent task options are effectively ignored for the remainder of the cycle. The best solution found in each cycle is saved if it is better than the best found in all the cycles thus far (or if it is the first cycle to initialize the best solution measures for later comparison), and the process is repeated for a maximum designated number of cycles; it does not terminate for stagnation to allow for potential better solutions (due to the ACO's stochastic edge selection capabilities). The best solution found is evaluated as described in Section 6.5.

Owing to the nature of the DLBP (i.e., whether or not a task should be added to a workstation depends on that workstation's idle time and precedence constraints), the probability in Formula 6.56 is not calculated once, at the beginning at each tour as is typical with ACO, but is calculated dynamically, generating new probabilities at each increment of t. The ACO probability calculation formula is modified for DLBP with the probability of ant r taking arc pq at time t during cycle NC calculated as

$$p_{pq}^{r}(t) = \begin{cases} \dfrac{[\tau_{pq}(\mathrm{NC})]^{\alpha} \cdot [\eta_{pq}(t)]^{\beta}}{\displaystyle\sum_{r \in \mathrm{allowed}_r} [\tau_{pr}(\mathrm{NC})]^{\alpha} \cdot [\eta_{pr}(t)]^{\beta}} & \text{if } q \in \text{allowed} \\ 0 & \text{otherwise} \end{cases} \tag{6.56}$$

Also modified for DLBP, the amount of trail on each arc is calculated after each cycle using

$$\tau_{pq}(\mathrm{NC}+1) = p \cdot \tau_{pq}(\mathrm{NC}) + \Delta\tau_{pq} \tag{6.57}$$

Where (unchanged from the general ACO formulation)

$$\Delta\tau_{pq} = \sum_{r=1}^{m} \Delta\tau_{pq}^{r} \tag{6.58}$$

and (also unchanged, other than the use of arcs versus edges and the recognition that arc $pq \neq qp$ in DLBP—further discussed at the end of this section)

$$\Delta_{\tau_{pq}^{r}} = \begin{cases} \dfrac{Q}{L_r} & \mathrm{arc}(p,q)\mathrm{used} \\ 0 & \mathrm{otherwise} \end{cases} \tag{6.59}$$

As in other DLBP combinatorial optimization methodologies, DLBP ACO seeks to preserve precedence while not exceeding CT in any workstation. As long as the balance is at least maintained, the DLBP ACO then seeks improvements in hazard measure, demand measure, and direction measure but never at the expense of precedence constraints. For DLBP ACO, the visibility, η_{pq}, is also calculated dynamically at each increment of t during each cycle and is defined as

$$\eta_{pq}(t) = \frac{1}{F_{tr}} \tag{6.60}$$

where F_{tr} is the balance of ant r at time t (i.e., ant r's balance thus far in its incomplete solution sequence generation). The divisor for the change in trail is defined for DLBP ACO as

$$L_r = F_{nr} \tag{6.61}$$

where a small final value for ant r's measure of balance at time n (i.e., at the end of the tour) provides a large measure of trail added to each arc. Though L_r and η_{pq} are related in this application, this is not unusual for ACO applications in general; for example, in the traveling salesman problem, L_r is the tour length while η_{pq} is the reciprocal of the distance between cities p and q [9]. However, this method of selecting η_{pq} (effectively a short-term greedy choice) may not always translate into the best long-term (i.e., final tour) solution for a complex problem like the DLBP. Also note that, although this is a multicriteria problem, only a single criterion (the measure of balance F) is being used in the basic ACO calculations and trail selection. The other objectives are only considered after balance and, only at the completion of each cycle, not as part of the probabilistic vertex selection. That is, an ant's tour solution is produced based on F, while at the end of each cycle the best overall solution is then updated based on F, H, D, and R in that order. This is done because the balance is considered (for the purpose of this study) to be the overriding

AQ4

requirement, as well as to be consistent with previous multicriteria DLBP studies by the authors. (One way to consider other criteria in the n greedy steps taken in the selection of a solution would be a weighting scheme in the probabilistic step selection.)

Procedure DLBP_*ACO *{

1. Initialize:
   SET NC := 0 {NC is the cycle counter}
   FOR every arc *pq* **SET** an initial value τ_{pq}(NC) := *c* for trail intensity and τ_{pq} := 0

2. Format problem space:
   SET *t* := 0 {*t* is the time counter}
   Place the *m* ants on the *n* vertices
   SET *s* := 1 {*s* is the Tabu list index}
   FOR *r* := 1 to *m* **DO**
   Place the starting part of the *r*th ant in **Tabu**$_r$(*s*)

3. Repeat until Tabu lists are full {this step will be repeated (*n* – 1) times}
   SET *s* := *s* + 1
   SET *t* := *t* + 1
   FOR *r* := 1 to *m* **DO**
   Choose the part *q* to move to, with probability $p^r_{pq}(t)$ as given by Formula (56) {at time *t* the *r*th ant is on part *p* = **Tabu**$_r$(*s*– 1)}
   Move the *r*th ant to the part *q*
   Insert part *q* in **Tabu**$_r$(*s*)

4. **FOR** *r* := 1 to *m* **DO**
   Move the *r*th ant from **Tabu**$_r$(*n*) to **Tabu**$_r$(1)
   Compute *F,H,D*, and *R* for the sequence described by the *r*th ant
   SAVE the best solution found per procedure BETTER_SOLUTION

   FOR every arc *pq*
   FOR *r* := 1 to *m* **DO**

$$\Delta\tau^r_{pq} := \begin{cases} \dfrac{Q}{L_r} & arc(p,q)\,used \\ 0 & otherwise \end{cases}$$

$$\Delta\tau_{pq} := \Delta\tau_{pq} + \Delta\tau^r_{pq}$$

5. **FOR** every arc *pq* compute τ_{pq}(NC + 1) according to τ_{pq}(NC + 1) := $\rho\cdot\tau_{pq}$(NC) + $\Delta\tau_{pq}$
   SET NC := NC + 1
   FOR every arc *pq* **SET** $\Delta\tau_{pq}$:= 0

6. **IF** (NC < NC_{max})
   THEN
   Empty all Tabu lists
   GOTO STEP 2
   ELSE
   PRINT *best_solution*
   STOP

FIGURE 6.10
DLBP ACO procedure.

Three procedural changes to the algorithm in Dorigo et al. [9] were made (Figure 6.10). First, the time counter was reset to zero in each cycle. This has no effect on the software's performance and was done only for program code readability. Second, "Place the *m* ants on the *n* nodes" was moved from step 1 to step 2 (and "nodes" was changed to "vertex" in recognition of DLBP's formulation as a digraph). This was done so that the resetting of each ant to each vertex would be repeated in each cycle, prohibiting the potential accumulation of ants on a single (or few) ending vertex (vertices) resulting in the subsequent restarting of all (or many) of the ants from that one (or few) vertex (vertices) in successive cycles; resetting the ants ensures solution diversity. This was also necessary due to the nature of the DLBP where the sequence is a critical and unique element of any solution. A part with numerous precedence constraints will typically be unable to be positioned in the front of the sequence (i.e., removed early) and will often be found toward the end. In DLBP, not resetting each ant could potentially result in a situation where, for example, all ants choose the same final part due to precedence constraints; when each ant attempts to initiate a subsequent search (i.e., the next cycle) from that last vertex, all fail due to the infeasibility of attempting to remove a final part first, effectively terminating the search in just one cycle. Finally, step 6 normally could also terminate for stagnation behavior, but (as discussed in the next section) this is not desired for DLBP and has been deleted.

Finally, it can be inferred from Ref. 9 that *pq* is equivalent to *qp*. Although this is acceptable and desirable in a TSP-type problem, in DLBP, sequence is an essential element of the solution (due to precedence constraints and size constraints at each workstation). For this reason, in DLBP ACO edges *pq* and *qp* are directed (i.e., arcs) and therefore distinct and unique and as such, when trail is added to one, it is not added to the other.

6.10.3.3 DLBP-Specific Quantitative Values

In DLBP ACO, the maximum number of cycles is set at 300 (i.e., $NC_{max} = 300$) since larger problems than those studied in this chapter were shown to reach their best solution by that count and as a result of experimentation by the authors. The process was not run until no improvements were shown but, as is the norm with many combinatorial optimization techniques, was run continuously on subsequent solutions until NC_{max} [23]. This also enabled the probabilistic component of ACO an opportunity to leave a potential local minimum. Repeating the DLBP ACO method in this way provides improved balance over time. As per the best ACO metaheuristic performance experimentally determined by Dorigo et al. [9], the weight of pheromone in path selection α was set equal to 1.00; the weight of balance in path selection β was set equal to 5.00; the evaporation rate, $1 - \rho$ was set to 0.50; and the amount of pheromone added if a path is selected Q was set to 100.00. The initial amount of pheromone on all of the paths c was set equal to 1.00.

6.10.3.4 DLBP ACO Numerical Results Using the 10-Part Case Study Instance

Over multiple runs, DLBP ACO was able to successfully find several (due to the stochastic nature of ACO) feasible solutions to the 10-part instance. Table 6.8 depicts a typical solution sequence. This solution demonstrates the minimum number of workstations while placing the hazardous part (part number 7) and high-demand parts (parts 6, 7, and 9) relatively early in the removal sequence (the exception being part 2, primarily due to precedence constraints and part 2 having a smaller part removal time that is not conducive to the ACO's iterative greedy process) and allowing just six direction changes (due to precedence constraints, the optimum is five while the worst case is eight). The speed for the C++ implemented program on this problem (averaged over five runs) was just over one-tenth of a second on the 1.6 GHz PM × 86-family workstation.

TABLE 6.8

Typical DLBP ACO Disassembly Sequence Solution to the 10-Part Data Set

		Workstation					
		1	2	3	4	5	
Part removal sequence	10	10					Time to remove part (in seconds)
	5	23					
	6		14				
	7		19				
	4			17			
	9			14			
	8				36		
	1					14	
	2					10	
	3					12	
Total time		33	33	31	36	36	
Idle time		7	7	9	4	4	

The optimality of the NWS minimization and the balancing is demonstrated by the solution consisting of all five workstations operating at 78–90% of capacity while meeting all precedence constraints in the exponentially large search space (i.e., 10! or 3,628,800).

6.10.4 The Greedy Algorithm and AEHC Heuristic

Two approaches are used sequentially to provide solutions to DLBP. The first rapidly provides a feasible solution to the DLBP and minimum or near-minimum NWS using a greedy algorithm. The greedy algorithm considered here is based on the first-fit-decreasing (FFD) algorithm (developed for the bin-packing problem and effectively used in computer processor scheduling). The second is implemented after the DLBP greedy algorithm to compensate for DLBP greedy's inability to balance the workstations. The AEHC heuristic ensures that the idle times at each workstation are similar. It does this by only comparing tasks assigned in adjacent workstations; this is done both to conserve search time and to only investigate swapping tasks that will most likely result in a feasible sequence.

AQ5

6.10.4.1 Greedy Model Description and the Algorithm

A greedy strategy always makes the choice that looks the best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. Greedy algorithms do not always yield optimal solutions but for many problems, they do [5]. The DLBP greedy algorithm [33] was built around FFD rules. FFD rules require looking at each element in a list, from largest to smallest (PRT in the DLBP) and putting that element into the first workstation in which it fits without violating precedence constraints. When all of the work elements have been assigned to a workstation, the process is complete. The greedy FFD algorithm is further modified with priority rules to meet multiple objectives. The hazardous parts are prioritized to the earliest workstations, greedy ranked large removal time to small. The remaining nonhazardous parts are greedy ranked next, large removal times to small. In addition, selecting the part with the larger demand ahead of those with lesser demands breaks any ties for parts with equal part removal times and selecting the part with an equivalent part removal direction breaks any ties for parts also having equal part removal directions. This is done to prevent damage to these more desirable parts. The DLBP greedy algorithm provides an optimal or near-optimal minimum number of workstations; the more constraints, the more likely the optimal number of workstations is found. The level of performance (minimal NWS) will generally improve with the number of precedence constraints.

The specific details for this implementation are as follows. The DLBP greedy algorithm first sorts the list of parts. The sorting is based on part removal times, whether or not the part contains hazardous materials, the

subsequent demand for the removed part, and the part removal direction. Hazardous parts are put at the front of the list for selection into the solution sequence. The hazardous parts are ranked from largest to smallest part removal times. The same is then done for the nonhazardous parts. Any ties (i.e., two parts with equal hazard typing and equal part removal times) are not randomly broken, but rather ordered based on the demand for the part, with the higher demand part being placed earlier on the list. Any of these parts also having equal demands then selected based on their part removal direction being the same as the previous part on the list (i.e., two parts compared during the sorting that only differ in part removal directions are swapped if they are removed in different directions—the hope being that subsequent parts and later sorts can better place parts having equal part removal directions). AQ6

Once the parts are sorted in this multicriteria manner, the parts are placed in workstations in FFD greedy order while preserving precedence. Each part in the sorted list is examined from first to last. If the part had not previously been put into the solution sequence (as described by ISS_k Tabu* list data structure), the part is put into the current workstation if idle time remains to accommodate it and as long as putting it into the sequence at that position will not violate any of its precedence constraints. ISS_k is defined as

$$ISS_k = \begin{cases} 1, & \text{assigned} \\ 0, & \text{otherwise} \end{cases} \tag{6.62}$$

If no workstation can accommodate it at the given time in the search due to precedence constraints, the part is maintained on the sorted list (i.e., its ISS_k value remains 0) and the next part (not yet selected) on the sorted list is considered. If all parts have been examined for insertion into the current workstation on the greedy solution list, a new workstation is created and the process is repeated. Figure 6.11 shows the DLBP greedy procedure.

While being very fast and generally very efficient, the FFD-based greedy algorithm is not always able to optimally minimize the number of workstations. In addition, there is no capability to balance the workstations; in fact, the FFD structure lends itself to filling the earlier workstations as much as possible, often to capacity, while later workstations end up with progressively greater and greater idle times. This results in an extremely poor balance. This limitation led to the formulation of follow-on methods to fulfill the second objective, one of which is AEHC (see McGovern and Gupta [33] for a 2-opt implementation modified for DLBP by exchanging vertices instead of edges as would be done in a TSP application).

* Although the name recalls Tabu search as proposed in Refs. 13,14, the ISS_k Tabu list is similar to that used by the ant system [9] including, for example, the absence of any aspiration function.

Procedure DLBP_GREEDY {

1. **SET** $j := 0$

2. $\forall$ $PRT_k \mid 1 \leq k \leq n$, generate sorted part sequence PSS based on:
   $h_k := 1$ parts (i.e., hazardous), large PRT_k to small; then $h_k := 0$ parts, large PRT_k to small; if more than one part has both the same hazard rating and the same PRT_k, then sort by demand, large d_k to small; if more than one part has the same hazard rating, PRT_k, and demand rating, then sort by equal part removal direction

3. $\forall$ $p \in$ PSS $\mid p := PSS_k$, $1 \leq k \leq n$, generate greedy part removal sequence PSG by:
   IF $ISS_p = = 1$ (i.e., part p already included in PSG)$\vee$
   PRECEDENCE_FAIL$\vee$
   $I_j < PRT_p$ (i.e., part removal time required exceeds idle time available at ST_j)

   THEN **IF** $p = = n$ (i.e., at last part in sequence)

   THEN Increment j (i.e., start new workstation)
   Start again at $p := 1$

   ELSE Try next p

   ELSE Assign p to ST_j and set $ISS_p := 1$

4. Using PSG:
   Calculate F, H, D, and R

}

FIGURE 6.11
DLBP greedy procedure.

6.10.4.2 Hill-Climbing Description and the Heuristic

A second-phase approach to DLBP was developed to quickly provide a near-optimal and feasible balance sequence using a hill-climbing local search heuristic, AEHC [30]. AEHC was tailored to DLBP (and applicable to any bin-packing-type problem having precedence constraints) to take advantage of knowledge about the problem's format and constraints to provide a solution that is better balanced than DLBP greedy alone but significantly faster than DLBP 2-opt [33]. Hill climbing is an iterated improvement algorithm, basically a gradient descent/ascent. It makes use of an iterative greedy strategy, which is to move in the direction of increasing value. A hill-climbing algorithm evaluates the successor states and keeps only the best one [23]. AEHC is designed to consider swapping each task in every workstation with each task only in the next adjacent workstation in search of improved balance. It does this while preserving precedence and not exceeding CT in any workstation. Only adjacent workstations are compared to enable a rapid search and since it is deemed unlikely that parts several workstations apart can be swapped and still preserve the precedence of all of the tasks in-between. As shown in the Figure 6.12 example, a part has a limited number of other parts it can be considered with for exchange. In the 2-opt methodology from McGovern and Gupta [33] and using the Figure 6.12 data, part number 6 would be considered for exchange by parts 1 through 5, and would consider

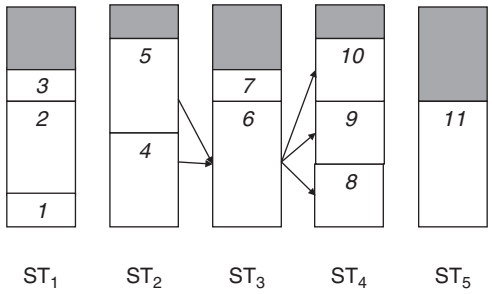

FIGURE 6.12
AEHC example.

exchanges with parts 7 through 11; that is, any part could be exchanged with any other part. In AEHC, part 6 would be considered for exchange only by parts 4 and 5, and would consider exchanges only with parts 8, 9, and 10.

The neighborhood definition and search details of AEHC are as follows. After the DLBP greedy algorithm generates a minimum-NWS, feasible solution, the AEHC heuristic is applied to improve the balance. AEHC does this by going through each task element (part) in each workstation and comparing it to each task element in the next adjacent workstation. If the two task elements can be exchanged while preserving precedence, without exceeding either workstations available idle time, and with a resulting improvement in the overall balance, the exchange is made and the resulting solution sequence is saved as the new solution sequence. This process is repeated until task elements of the last workstation have been examined. The heuristic is given in pseudocode format in Figure 6.13.

In the DLBP greedy/AEHC hybrid heuristic, the DLBP greedy process is run once to generate an initial solution. Hill climbing is typically continuously run on subsequent solutions for as long as is deemed appropriate or acceptable by the user or until it is no longer possible to improve, at which point it is assumed that the local optima has been reached [23]. Repeating the AEHC method in this way provides improved balance with each iteration. The AEHC was tested both ways; run only once after the greedy solution was generated, as well as run until the local optima was obtained (a single AEHC iteration has several benefits including the observation that AEHC was seen to typically provide its largest single balance performance improvement in the first iteration). Additionally, a single iteration of AEHC may be recommended for the real-time solution of a very large DLBP instance on a dynamic mixed-model and mixed-product disassembly line.

6.10.4.3 DLBP Greedy/AEHC Hybrid Heuristic Numerical Results Using the Cell Phone Case Study Instance

Table 6.9 shows the solution generated by the DLBP greedy/AEHC hybrid heuristic; shading indicates workstation assignment. As per its design, DLBP

Procedure* DLBP_AEHC *{

1. Initialize:
   SET PST := PSG
   SET TMP := PSG
   SET BST := PSG
   SET *START* := 1

2. $\forall\, j \mid 1 \le j \le \text{NWS} - 1$, attempt to generate better balanced AEHC part removal sequence PST by performing AEHC part swap (i.e., swap each TMP_p in workstation j with every TMP_q in workstation $j + 1$)

   $\forall\, p \in \text{TMP} \mid p = \text{TMP}_k, START \le k \le START + \text{NPW}_j - 1$

   $\forall\, q \in \text{TMP} \mid q = \text{TMP}_k, START + \text{NPW}_j \le k \le START + \text{NPW}_j - 1 + \text{NPW}_{j+1}$

   Calculate new ST_j, and ST_{j+1}

   IF ((CT – new $\text{ST}_j \ge 0$) $\wedge$
   (CT – new $\text{ST}_{j+1} \ge 0$))

   THEN Calculate new *F*, *H*, *D*, and *R*

   IF ((BETTER_SOLUTION(TMP, BST))
   $\wedge$
   (PRECEDENCE_PASS))

   THEN SET BST := TMP

   SET *START* := *START* + NPW_j

3. Save results:
   SET PST := BST
   SET TMP := BST

4. Repeat STEP 2 until no more improvements in *F*, *H*, *D*, or *R* can be made

}

FIGURE 6.13
DLBP AEHC procedure.

TABLE 6.9

DLBP Greedy/AEHC Hybrid Heuristic Solution to the Cell Phone Instance

Part ID	1	2	4	3	6	7	8	9	16	5	10	15	18	14	13	17	20	11	12	21	25	19	22	23	24
PRT	3	2	10	3	15	15	15	15	2	10	2	2	3	2	2	2	5	2	2	1	2	18	5	15	2
Workstation	1	1	1	1	2	3	4	5	5	6	6	6	6	7	7	7	7	7	7	7	7	8	9	10	10
Hazardous	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0
Demand	4	7	1	1	1	1	1	1	1	1	2	1	2	1	1	2	1	1	4	4	4	8	6	7	1
Direction	2	3	5	5	5	5	5	5	2	5	4	0	5	2	5	4	4	4	4	4	4	5	4	4	4

greedy removed hazardous parts early on, though many of these parts had to wait due to precedence constraints. Four parts with larger part removal times (15 s) were removed very early on (although parts 19 and 23 take as long or longer, precedence constraints prevent earlier removal) but at the expense

of using an entire workstation, demonstrating a condition that limits DLBP greedy/AEHC's effectiveness since the greedy portion works to take out large (time) items early (and therefore, potentially adjacent to each other in the sequence) and AEHC is not allowed to look out far enough to add smaller time items into the sequence to better fill those workstations. As can be seen in Figure 6.6, precedence relationships play a large role in determining the allowable solution sequences to this problem.

The DLBP greedy/AEHC heuristic hybrid was run three times to obtain an average computation time (due to the hybrid's deterministic nature, all runs generated the same solution sequence). The solution to the cell phone instance was found extremely quickly, regularly taking less than 1/100th of a second. The hybrid technique was not, however, able to generate the optimal number of workstations or the optimal balance (with the assumption that NWS = 9 and $F = 9$ is optimal [37]).

6.10.5 The H–K General-Purpose Heuristic

A new search approach has been proposed to provide a rapid, near-optimal search for combinatorial optimization applications. This heuristic rapidly provides a feasible solution using a modified exhaustive search technique to provide a data sampling of all solutions.

6.10.5.1 Heuristic Background and Motivation

Exhaustive search is optimal because it looks at every possible answer. In many physical search applications (e.g., antisubmarine warfare, search, and rescue), exhaustive search is not possible due to time or sensor limitations. In these cases, it becomes practical to sample the search space and operate under the assumption that, for example, the highest point of land found in a limited search is either the highest point in a given search area or is reasonably near the highest point. The proposed search technique [31,41] in this section works by sampling the exhaustive solution set; that is, search the solution space in a method similar to an exhaustive search, but in a pattern that skips solutions (conceptually similar to the STEP functionality in a FOR loop as found in computer programming) to significantly minimize the search space (in Figure 6.14, the shading indicates solutions visited, the border represents the search space).

This pattern is analogous to the radar acquisition search pattern known as "spiral scan," the search and rescue pattern of the "expanding square," or the antisubmarine warfare aircraft "magnetic anomaly detector (MAD) hunting circle." Once the solution is generated, the space can be further searched with additional application of the H–K metaheuristic (with modifications from the previous H–K; see the end of Section 6.10.5.2), or the best-to-date solution can be further refined by performing subsequent local searches (such as 2-opt or smaller, localized H–K searches). That is, depending on the application, H–K can be run once, multiple times on subsequent solutions,

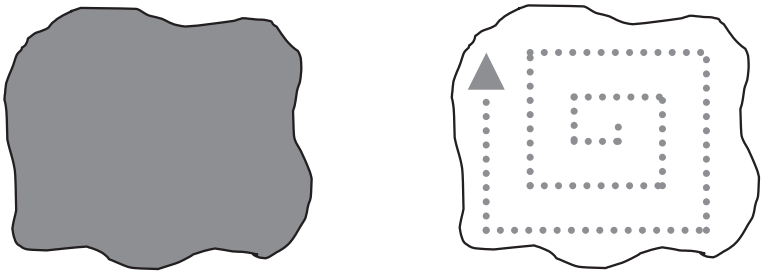

FIGURE 6.14
Exhaustive search space and the H–K search space and methodology.

multiple times from the same starting point using different skip measure (potentially as a multiprocessor or grid computing application), multiple times from a different starting point using the same skip measure (again, potentially as a multiprocessor or grid computing application), or followed up with an H–K or another, differing local search on the best or several of the best suboptimal solutions generated. H–K can also be used as the first phase of a hybrid algorithm or to hot start another methodology (e.g., to provide the initial population in a GA). For the purpose of demonstrating the method and measuring its efficacy, H–K is run alone to a single-phase solution in this chapter. The skip size ψ can be as small as $\psi = 1$, or as large as $\psi = n$. Since $\psi = 1$ is equivalent to exhaustive search, and $\psi = n$ generates a trivial solution (it returns only one solution, that being the data in the same sequence as it is given to H–K, i.e., $PS_k = \langle 1, 2, 3, \ldots, n \rangle$; also, in the single-phase H–K, this solution is already considered by any value of ψ), in general all skip values can be further constrained as

$$2 \leq \psi_k \leq n - 1 \tag{6.63}$$

6.10.5.2 H–K Methodology Comparison to Other Solution Generating Methodologies

The H–K general-purpose heuristic shows a variety of similarities to many currently accepted combinatorial optimization methodologies. For example, like ACO and GA—but unlike Tabu search—H–K generates the best solutions when suboptimal solutions bear a resemblance to the optimal solution. In three dimensions, this would appear as a set with relatively shallow gradients (i.e., no sharp spikes); the more shallow the gradient, the better the chance the found solution is near the optimal solution. These data sets are effective in ACO applications since the ant agents work to reinforce good solution sections by adding trail (similar to pheromones in actual ants). They are effective in GA applications since those algorithms break apart good solutions and recombine them with other good solution sections, again, reinforcing preferred tours. The drawback to these is that if suboptimal solutions do not bear a resemblance to the optimal solution, the optimal solution may not be found. However, in many applications, it is not typical for the data

to perform in this manner. Also, it is not a significant drawback since the general H–K takes a deterministic, nonweighted path; therefore, an isolated solution is as likely to be visited as the optimal shallow gradient solution. Isolated or steep gradient solutions are addressed in ACO and GA with probabilistically selected tours, allowing for potential excursions to seemingly poor-performing areas of the search space.

Like simulated annealing (SA)—and unlike ACO, GA, and Tabu—the multiple phase version of H–K looks at a large area initially, then smaller and smaller areas in more detail.

Like Tabu search (and unlike ACO and GA), the single phase of H–K does not revisit solutions found previously during the same phase of search.

Unlike many applications of ACO, Tabu, and SA, (but like GA) H–K does not generate solutions similar to a branch-and-bound type of method where the best solution is grown by making decisions on which branch to take as the sequence moves from beginning to end. Rather, an entire solution sequence is generated prior to evaluation. Similarly, H–K does not contain any greedy component since the movement through the search space is deterministic and no branching decisions are required, which are typically made based upon the best short-term (greedy) move.

As with each of these heuristics, H–K can be applied to a wide range of combinatorial optimization problems. Like greedy and *r*-opt [29] heuristics and traditional mathematical programming techniques such as linear programming, H–K selects solutions for consideration based on a deterministic manner, while metaheuristics such as ACO and GA have a strong probabilistic component. As a result, H–K is repeatable, providing the same solution every time it is run. However, probabilistic components can be added in the form of randomized starting point(s), skip size(s), or skip types.

Unlike traditional mathematical programming techniques such as linear programming, H–K is not limited to linear problem descriptions and, as shown in this chapter using the DLBP, readily lends itself to multicriteria decision making as well as nonlinear problem formats. However, like most heuristics, it cannot consistently provide the optimal solution, while most mathematical programming methods generally do. Also common to heuristics, there is some degree of tuning and design required by the user, for example, selection of number of generations, mutation rate, crossover rate and crossover method in GA, or pheromone evaporation rate, number of ants, number of cycles and visibility description for ACO. H–K decisions include:

- *Number of H–K processes*. One or multiple
- *Starting point*. Constant, deterministic varying (i.e., different but known starting points for use multiple H–K processes only), random
- *Skip type* (*type of skipping between solution elements*). Constant type, deterministic varying type, random type

- *Skip size* (*size of data skipped in each solution element*). Constant size, deterministic varying size, random size
- *Follow-on solution refinement*. None, different H–K (different starting point, skip type, skip size, etc.), local H–K (H–K search about previous solution), other (such as 2-Opt or GA).

Depending on these structural decisions, H–K can take on a variety of forms; from a classical optimization algorithm in its most basic form, to a general evolutionary algorithm (EA) with the use of multiple H–K processes, to a biological or natural process algorithm by electing random functionality. To demonstrate the method and show some of its limitations, in this chapter the most basic form of the H–K metaheuristic is used; one process, constant starting point of $PS_k = \langle 1, 1, 1, \ldots, 1\rangle$ (since the solution set is a permutation, there are no repeated items, therefore the starting point is effectively $PS_k = \langle 1, 2, 3, \ldots, n\rangle$), constant skip type (i.e., each element in the solution sequence is skipped in the same way), constant skip size of ψ (although different skip sizes are used throughout the chapter to allow for a thorough efficacy analysis, the skip size stays constant throughout each H–K run), and no follow-on solution refinement.

6.10.5.3 The H–K Process and DLBP Application

As far as the H–K process itself, since it is a modified exhaustive search allowing for solution sampling, it searches for solutions similar to depth-first search iteratively seeking the next permutation iteration—allowing for skips in the sequence—in lexicographic order. In the basic H–K and with $\psi = 2$, the first element in the first solution would be 1, the next element considered would be 1, but since it is already in the solution, that element would be incremented and 2 would be considered and be acceptable. This is repeated for all of the elements until the first solution is generated. In the next iteration, the initial part under consideration would be incremented by 2 and, therefore, 3 would be considered and inserted as the first element. Since 1 is not yet in the sequence, it would be placed in the second position, 2 in the third, etc. For DLBP H–K, this is further modified to test the proposed sequence part addition for precedence constraints. If all possible parts for a given solution position fail these checks, the remainder of the positions are not further inspected, the procedure falls back to the previously successful solution addition, increases it by 1, and continues. These processes are repeated until all allowed items have been visited in the first solution position (and by default, due to the nested nature of the search, all subsequent solution positions). For example, with $n = 4$, $P = \{1, 2, 3, 4\}$, and no precedence constraints, instead of considering the $4! = 24$ possible permutations, only five are considered by the single-phase H–K with $\psi = 2$ and using forward-only data: $PS_k = \langle 1, 2, 3, 4\rangle$; $PS_k = \langle 1, 4, 2, 3\rangle$; $PS_k = \langle 3, 1, 2, 4\rangle$; AQ7

$PS_k = \langle 3, 1, 4, 2 \rangle$; and $PS_k = \langle 3, 4, 1, 2 \rangle$. All of the parts are maintained in a Tabu-type list ISS_k defined by Formula 6.62. Each iteration of the DLBP H–K generated solution is considered for feasibility. If it is ultimately feasible in its entirety, DLBP H–K then looks at each element in the solution and places that element using the next-fit (NF) rule (from the bin-packing problem application; once a bin has no space for a given item attempted to be packed into it, that bin is never used again even though a later, smaller item may appear in the list and could fit in the bin [24]). DLBP H–K puts the element under consideration into the current workstation if it fits. If it does not fit, a new workstation is assigned and previous workstations are never again considered. Although NF does not perform as well as first-fit, best-fit, first-fit-decreasing, or best-fit-decreasing when used in the general bin-packing problem, it is only one of these rules that will work with a DLBP solution sequence due to the existence of precedence constraints (see Section 6.10.4 or McGovern and Gupta [33] for a DLBP implementation of First-Fit-Decreasing). When all of the work elements have been assigned to a workstation, the process is complete and the balance, hazard, demand, and direction measures are calculated. The best of all of the inspected solution sequences is then saved as the problem solution. Although the actual software implementation for this dissertation consisted of a very compact recursive algorithm, in the interest of clarity, the general DLBP H–K procedure is presented here as a series of nested loops (Figure 6.15).

The following section studies how the skip size affects various measures including time complexity. The general form of the skip size-to-problem size relationship is formulated as

$$\psi_k = n - \Delta\psi_k \tag{6.64}$$

While not described in Section 6.10.5.1, early tests of time complexity growth with skip size suggest another technique to be used as part of H–K search. Since any values of ψ that are larger than the chosen skip value for a given H–K problem take significantly less processing time, considering all larger skip values should also be considered to increase the search space at the expense of a minimal increase in search time. In other words, H–K can be run repeatedly on a given instance using all skip values from a smallest ψ (selected based on time complexity considerations) to the largest (i.e., $n - 1$ as per Formula 6.63) without a significant time penalty. In this case, any ψ_k would be constrained as

$$n - \Delta\psi_k \leq \psi_k \leq n - 1 \quad \text{where } 1 \leq \Delta\psi_k \leq n - 2 \tag{6.65}$$

If this technique is used (as it is in the next sections), it should also be noted that multiples of ψ visit the same solutions; for example, for $n = 12$ and

```
Procedure DLBP_H–K {
    SET ISS_k := 0 ∀ k ∈ P
    SET FS := 1
    PS_1 := 1 to n, skip by ψ_1
        SET ISS_PS1 := 1
        PS_2 := 1 to n, skip by ψ_2
            WHILE (ISS_PS2 = = 1 ∨
                   PRECEDENCE_FAIL∧
                   not at n)
                Increment PS_2 by 1
            IF ISS_PS2 = = 1
            THEN SET FS := 0
            ELSE SET ISS_PS2 := 1
                :
                :
            IF FS = =1
                PS_n := 1 to n skip by ψ_n
                    WHILE (ISS_PSn = = 1 ∨
                           PRECEDENCE_FAIL ∧
                           not at n)
                        Increment PS_n by 1
                    IF ISS_PSn = = 0
                    THEN evaluate solution PS
                :
                :
            IF FS = = 1
            THEN SET ISS_PS2 := 0
            ELSE SET FS :=1
        SET ISS_PS1 := 0
        SET FS :=1
}
```

FIGURE 6.15
DLBP H–K procedure.

$2 \leq \psi \leq 10$, the four solutions considered by $\psi = 10$ are also visited by $\psi = 2$ and $\psi = 5$.

6.10.5.4 DLBP A Priori Numerical Results for Varying Skip Size

The benchmark data set from Section 6.9.4 was then used with DLBP H–K to provide a thorough efficacy analysis. This section also demonstrates proper use of the data set for complexity studies, efficacy analysis, and graphical representation, as well as an overview of the findings with regard to the H–K general-purpose heuristic. The final configuration of the developed benchmark was: $|\text{PRT}| = 4$, 19 instances with instance size evenly distributed from $n = 8$ to $n = 80$ in steps of $|\text{PRT}|$. This provided numerous instances of predetermined, calculable solutions with the largest

instance 10 times larger than the smallest instance. The size and range of the instances is considered appropriate and significant for this study, with small *n*s tested—which decreases the NWS and tends to exaggerate less than optimal performance—as well as large, which demonstrates time complexity growth and efficacy changes with *n*. To summarize, the test data consisted of $8 \leq n \leq 80$ parts with 4 unique part removal times giving PRT = {3, 5, 7, 11}. Only the last part having PRT = 11 is hazardous; only the last part having PRT = 7 is demanded. The first of each part with PRTs equal to 3, 5, 7, and 11 are removed in direction $+x$; all others are in direction $-x$. The disassembly line is paced and operated at a speed that allows 26 s (CT = 26) for each workstation. Known optimal results include $F^* = 0$, $H^* = 1$, $D^* = 2$, $R^* = 1$.

The DLBP *a priori* instances were first used to determine how skip size affects performance and time complexity then; based on these results, it was used to determine how the heuristic's performance changes with problem size. Because H–K is exceptionally deterministic and performs no preprocessing of the data, the order in which the data are presented can affect the solutions considered. For this reason, the analyses done in the following sections were first run with the data as given by Formula 6.48 and then run again, but this time with the data presented in reverse. The better of the two results were used and the search times (forward and reverse) were added. Both forward and reverse were used to ensure unusually good (or bad) results that were not artificially obtained due to some unknown structural feature of the DLBP *a priori* data sets, since presenting data in both forward and reverse formats would be expected to be the most common application. This is not necessary where an instance contains a relatively large number of precedence constraints, individual parts in an instance do not bear a great deal of resemblance to each other, or where in an actual application one may expect a user of H–K to only run the data exactly as it was presented to them. Although not covered in this chapter, since this is an introduction to the basic concepts of H–K, note that it could be expected that better results may be achieved with multiple runs of H–K consisting of the data being presented to H–K in differing orders. (This is an effective H–K technique that is not described in Section 6.10.5.1, but one which may be the easiest to implement.)

Skip size was varied to determine if and how solution performance decreased with increases in ψ. All cases are the DLBP *a priori* data set with $n = 12$, and consideration of all skip sizes (including exhaustive search); that is, $\psi = \{1, \ldots, n\}$. This problem size was chosen because it is the largest multiple of |PRT| that could be repeatedly evaluated by exhaustive search in a reasonable amount of time ($n = 12$ took just over 20 min, while $n = 16$ is calculated to take over 2 years). This was essential since exhaustive search provides the time complexity and solution performance baseline. A full range of skip sizes was used to provide maximum and minimum performance measures, and all intermediate skip sizes were used to allow the highest level of detail for the study. Although most of the following results are not

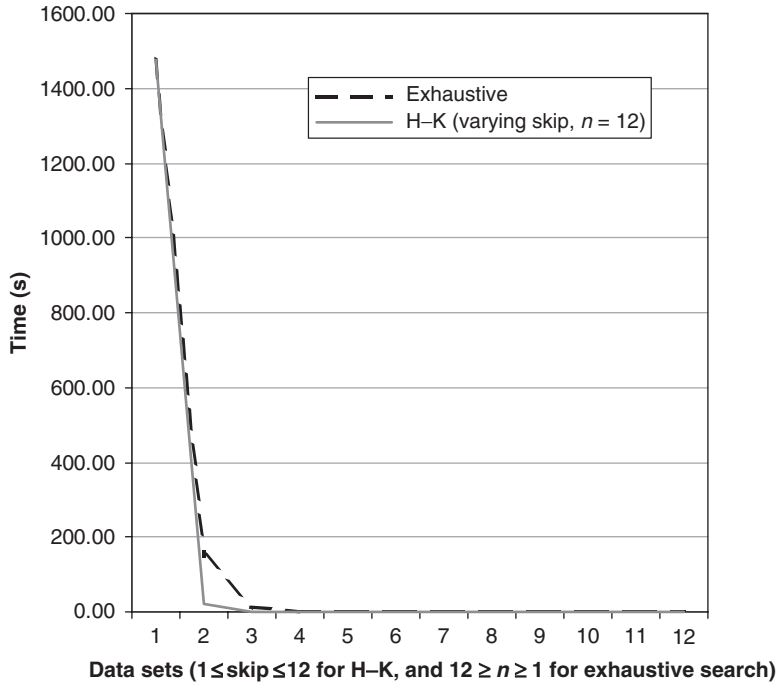

FIGURE 6.16
DLBP H–K time complexity.

optimal, although there exist multiple optimum extreme points, this is more of a reflection of the specially designed DLBP *a priori* data set. Suboptimal solutions are not an atypical result when this data set is evaluated. The data are intended to pose challenges, even at relatively small n values, to a variety of combinatorial optimization techniques.

The first analysis done was a time complexity study. As shown in Figure 6.16, the solution time appears to grow slightly less than factorially to the inverse of ψ, so even a very small change in skip size will yield a significant speed up of any search. (The exhaustive search curve has no relationship to the H–K curve and is depicted only to provide a size and shape reference.) A more detailed view of the H–K time-to-skip size relationship can be seen in Figure 6.17. With an exponential curve (n^n) adjusted for scale and overlaid, it can be seen that the rate of growth is actually exponential to the inverse of ψ.

Next, the balance performance was measured. The measure of balance was seen to remain optimal through $\psi = 4$ (i.e., while ψ stayed within 1/3 of n). As seen in Figure 6.18, performance then decreases as skip size increases but at a relatively slow rate.

The third objective was also seen to decrease in performance with increases in skip size; see Figure 6.19. The hazard measure remains optimal through $\psi = 3$, but eventually drops to worst case at $\psi = 9$.

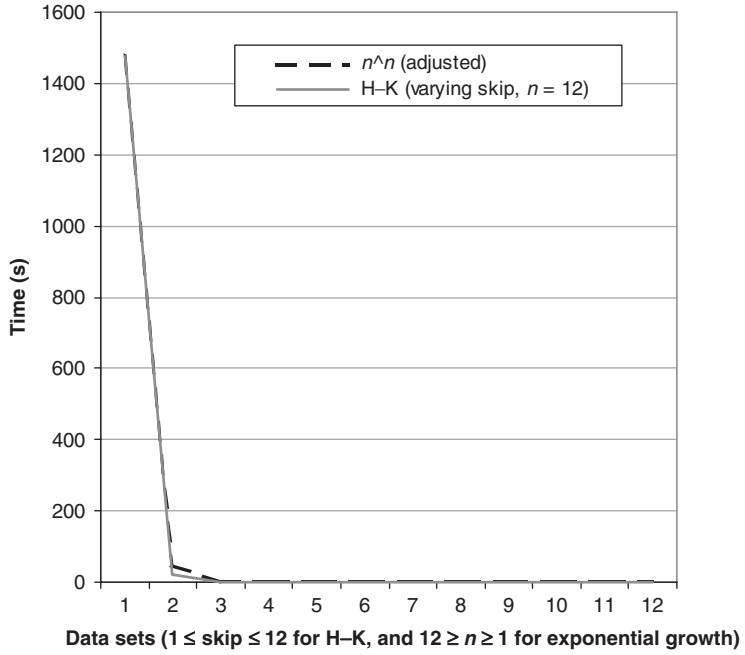

FIGURE 6.17
Detailed DLBP H–K time complexity.

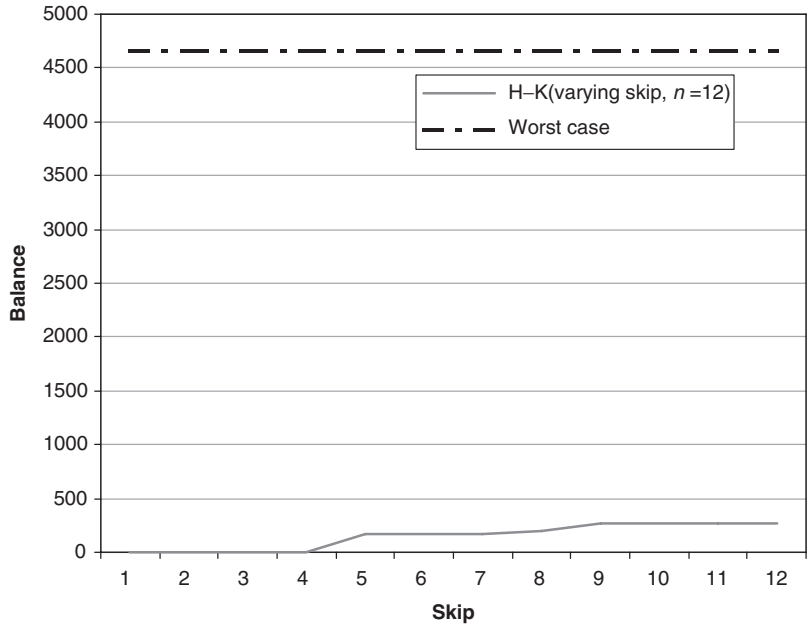

FIGURE 6.18
DLBP H–K F performance with changes in ψ.

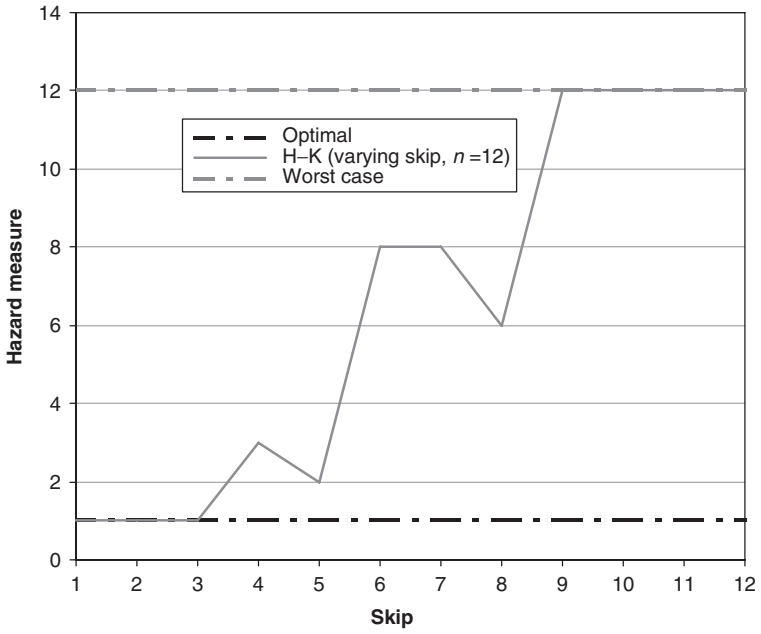

FIGURE 6.19
DLBP H–K *H* performance with changes in ψ.

The fourth and fifth objectives decreased in performance with increases in skip size as well, each at a slightly steeper angle (attributed to the prioritization of the objectives); see Figures 6.20 and 6.21. Optimal and better than optimal (i.e., removing the high-demand part first in the situation where the hazardous part is not removed first) *D* results were seen as far out as $\psi = 8$. The part removal direction measure reaches worst case by $\psi = 6$, but then improves slightly at $\psi = 9$.

These studies show how the heuristic's performance decreases with skip size and should provide at least a starting point for the selection of skip size based on the required level of performance as considered against time complexity. In large problems, time will quickly become the overriding consideration. These and other experiments indicate that a $\Delta\psi$ ranging from 10 to 12 (resulting in a ψ equal to from 10 to 12 less than n) gives time complexity performance on par with many other metaheuristic techniques, regardless of hardware implementation.

6.10.5.5 DLBP A Priori Numerical Results for Varying n

Problem size was varied to determine if and how solution performance changed with increases in n. All cases are the DLBP *a priori* problem with size varying between $8 \leq n \leq 80$, $1 \leq \Delta\psi \leq 10$, and the resulting skip sizes of $n - 10 \leq \psi \leq n - 1$ based on the work of the previous section. On the full

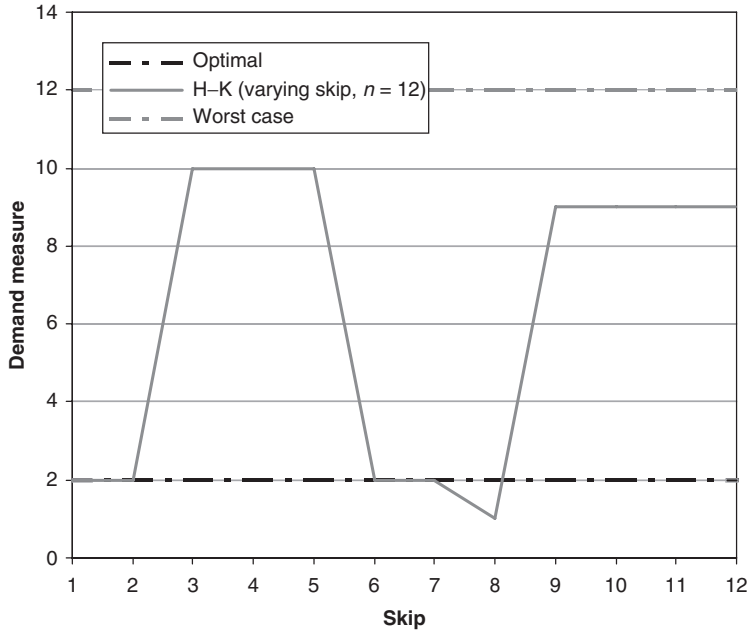

FIGURE 6.20
DLBP H–K *D* performance with changes in ψ.

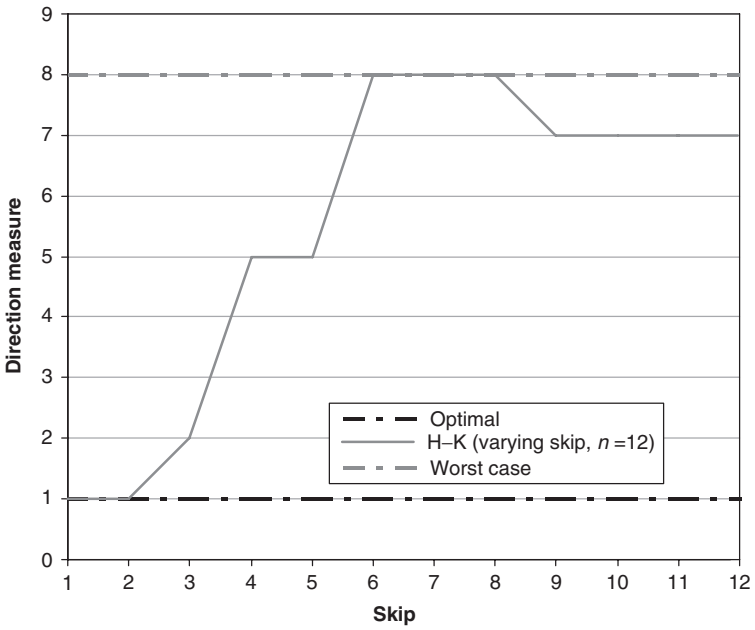

FIGURE 6.21
DLBP H–K *R* performance with changes in ψ.

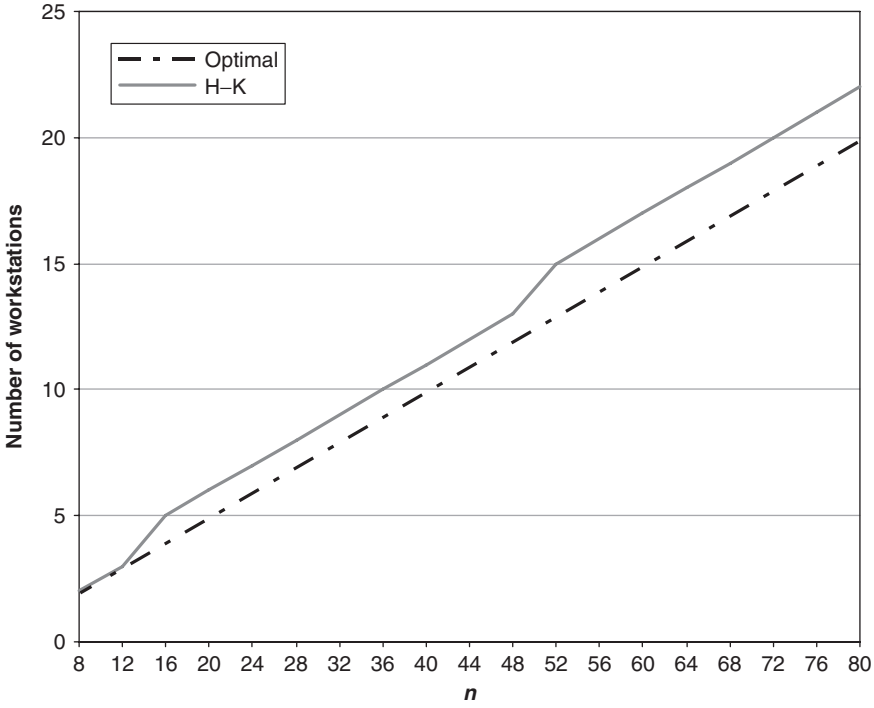

FIGURE 6.22
DLBP H–K workstation calculation.

range of data ($8 \leq n \leq 80$), DLBP H–K found solutions with NWS* workstations up to $n = 12$, then solutions with NWS* + 1 workstations through data set 11 ($n = 48$), after which it stabilized at NWS* + 2 (Figure 6.22). Increases in the balance measure are seen with increases in data set size. A detailed view of balance performance with problem size can be seen in Figure 6.23.

The hazardous part and the demanded part were both regularly suboptimally placed. Hazard part placement stayed relatively consistent with problem size (though effectively improving as compared to the worst case; see Figure 6.24), although demand decreased in performance when compared to best case and worst case (see Figure 6.25). These results are as expected because hazard performance is designed to be deferential to balance and is affected only when a better hazard measure can be attained without adversely affecting balance, although demand performance is designed to be deferential to balance and hazardous-part placement and is affected only when a better demand measure can be attained without adversely affecting balance or hazardous-part placement.

With part removal direction structured as to be deferential to balance, hazard, and demand, it was seen to decrease in performance when compared

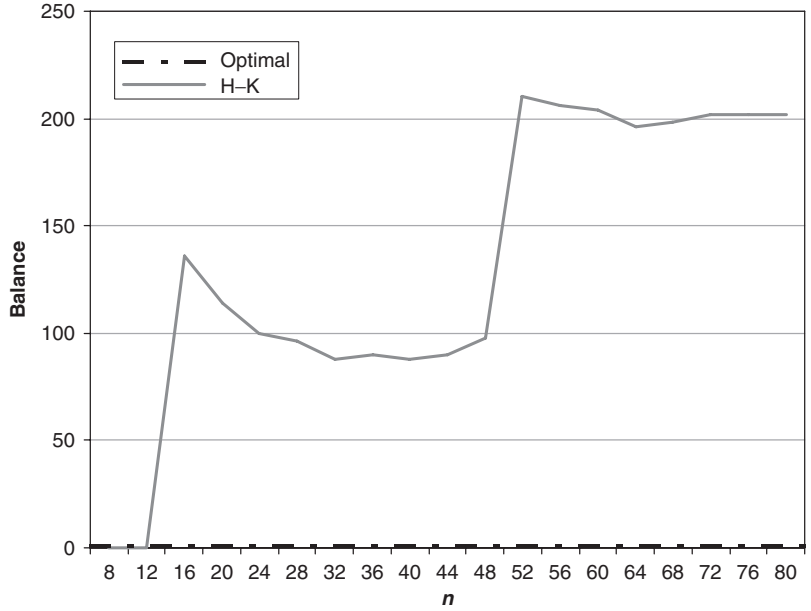

FIGURE 6.23
Detailed DLBP H–K balance measure.

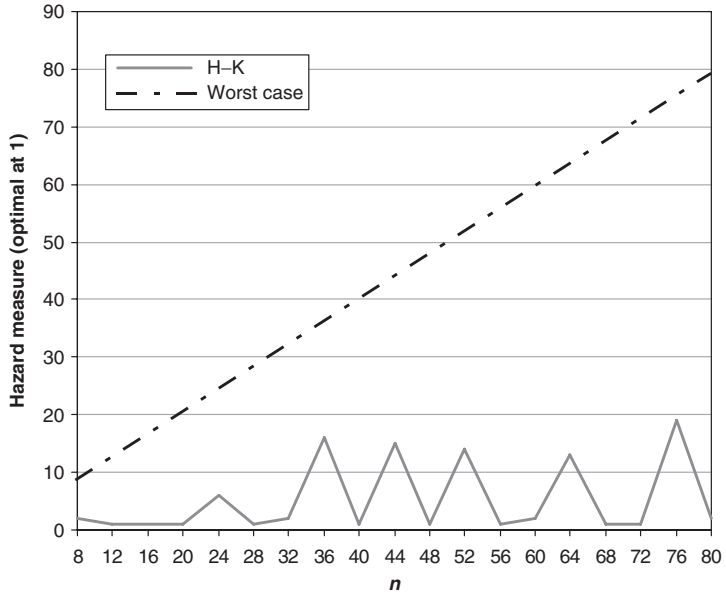

FIGURE 6.24
DLBP H–K hazard measure of performance.

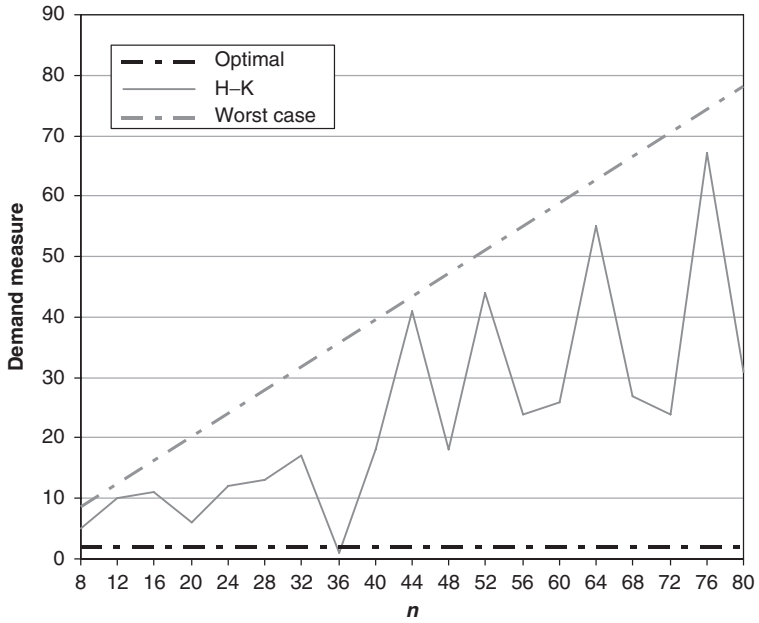

FIGURE 6.25
DLBP H–K demand measure of performance.

to the best case and when compared to the worst case (see Figure 6.26). Again, these results are as expected due the prioritization of the multiple objectives.

Although less than optimal, these results are not unusual for heuristics run against this data set. The DLBP *a priori* benchmark data is especially designed to challenge the solution-finding ability of a variety of search methods to enable a thorough quantitative evaluation of these methods' performance in different areas. A smaller ψ or, as with other search techniques, the inclusion of precedence constraints will increasingly move the DLBP H–K method toward the optimal solution. As shown in Figures 6.27 and 6.28, the time complexity performance of DLBP H–K provides the tradeoff benefit with the technique's near-optimal performance, demonstrating the moderate increase in time required with problem size that grows markedly slower than the exponential growth of exhaustive search. Exhaustive, n^2 and n^3 curves are shown for comparison. (The anomaly seen in the H–K curve in Figure 6.27 is due to a software rule added by the authors that dictated all ψ be equal to $n - 10$, but no less than $\psi = 3$, to prevent exhaustive or near-exhaustive searches at small n.)

Average-case time complexity using DLBP *a priori* data then appears to be $O(b^b)$ in skip size (where $b = 1/\psi$), while H–K appears to grow at approximately $O(n^2)$ in problem size.

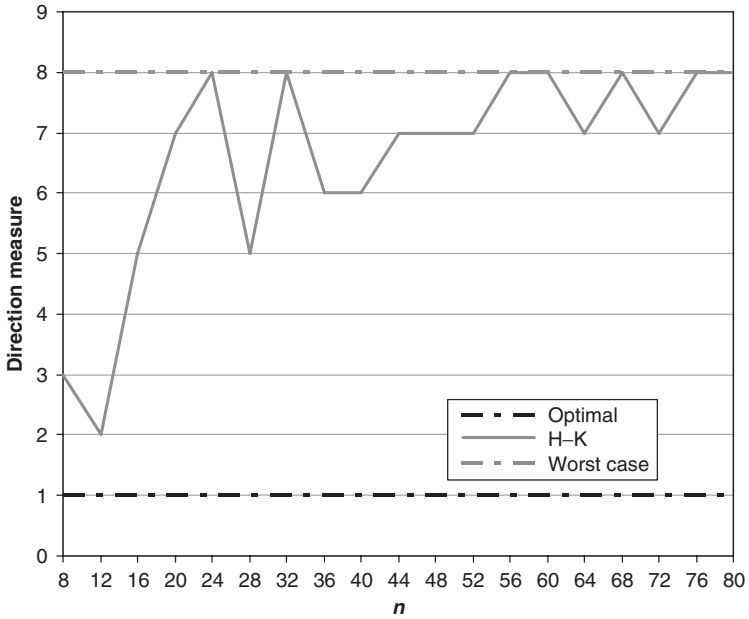

FIGURE 6.26
DLBP H–K part removal direction measure of performance.

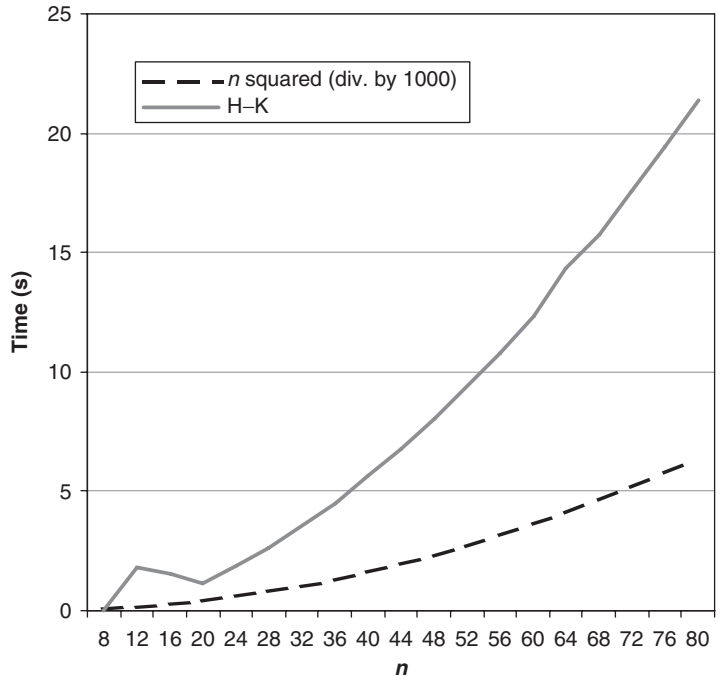

FIGURE 6.27
Detailed DLBP H–K time growth with problem size.

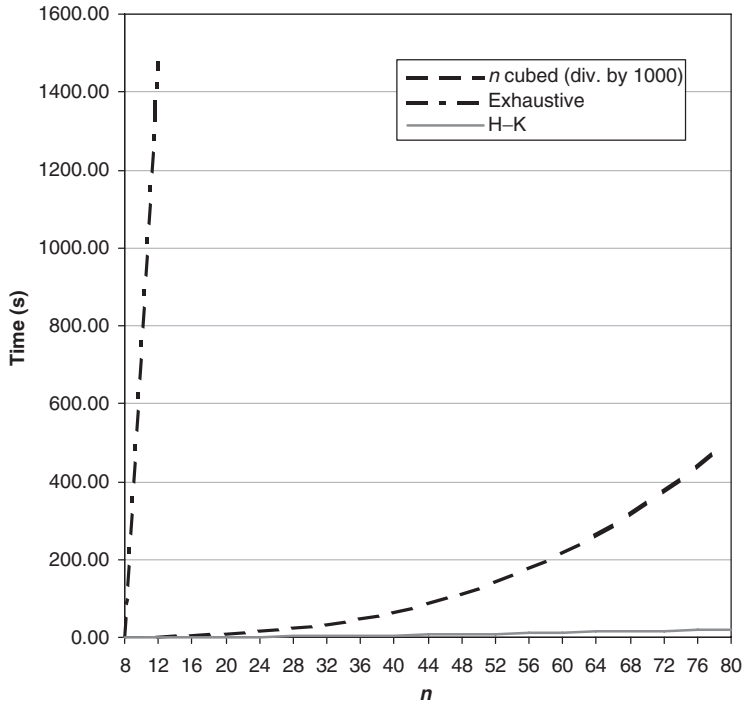

FIGURE 6.28
DLBP H–K time complexity compared to exhaustive search.

6.11 Conclusions

The disassembly line balancing problem was qualitatively introduced and mathematically defined, then proven to be unary NP-complete. Four very fast, near-optimal combinatorial optimization approaches to the problem were presented along with the four primary instances. The heuristics used rapidly provided a feasible solution to the DLBP using a wide variety of search techniques. Each demonstrates a near-optimal minimum number of workstations, with the level of optimality increasing with the number of constraints. They are also seen to generate feasible sequences with an optimal or near-optimal McGovern–Gupta measure of balance, while maintaining or improving the hazardous-materials measure, the demand measure, and the part removal direction measure. Although near-optimum methods, the DLBP versions of the ACO and GA metaheuristics, and the H–K general-purpose heuristic, along with a purpose-designed greedy/hill-climbing hybrid quickly found near-optimal solutions in DLBP's exponentially large search space. These combinatorial optimization methods are well suited to the multicriteria decision-making problem format as well as for the solution

of problems with nonlinear objectives. In addition, they are ideally suited to integer problems, a requirement of many disassembly problems, which generally do not lend themselves to rapid or easy solution by traditional optimum solution-generating mathematical programming techniques.

References

1. Aho, A. V., Hopcroft, J. E. and Ullman, J. D., *The Design and Analysis of Computer Programs*, Addison-Wesley, Reading, MA, 1974.
2. Bautista, J. and Pereira, J., Ant algorithms for assembly line balancing, in Dorigo, M. et al. (Eds.), *ANTS 2002, LNCS 2463*, Springer, Berlin, pp. 65–75, 2002.
3. Bierwirth, C., Mattfeld, D. C. and Kopfer, H., On permutation representations for scheduling problems, parallel problem solving from nature, in Voigt, H. M., Ebeling, W., Rechenberg, I. and Schwefel, H. P. (Eds.), *Lecture Notes in Computer Science*, Springer, Berlin, Vol 1141, pp. 3.10–3.18, 1996.
4. Brennan, L., Gupta, S. M. and Taleb, K. N., Operations planning issues in an assembly/disassembly environment, *International Journal of Operations and Production Planning*, 14, 9, 57–67, 1994.
5. Cormen, T., Leiserson, C., Rivest, R. and Stein, C., *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 2001.
6. Dantzig, G. B., Discrete-variable extremum problems, *Operations Research*, 5, 266–277, 1957.
7. Dorigo, M. and Di Caro, G., The ant colony optimization meta-heuristic, in Corne, D., Dorigo, M. and Glover, F. (Eds.), *New Ideas in Optimization*, McGraw-Hill, Maidenhead, UK, pp. 11–32, 1999.
8. Dorigo, M., Di Caro, G. and Gambardella, L. M., Ant algorithms for discrete optimization, *Artificial Life*, 5, 3, 137–172, 1999.
9. Dorigo, M., Maniezzo, V. and Colorni, A., The ant system: optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 26, 1, 1–13, 1996.
10. Elsayed, E. A. and Boucher, T. O. *Analysis and Control of Production Systems*, Prentice Hall, Upper Saddle River, NJ, 1994.
11. Erel, E. and Gokcen, H., Shortest-route formulation of mixed-model assembly line balancing problem, *Management Science*, 11, 2, 308–315, 1964.
12. Garey, M. and Johnson, D., *Computers and Intractability: A Guide to the Theory of NP Completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.
13. Glover, F., Tabu search—part I, *ORSA Journal on Computing*, 1, 3, 190–206, 1989.
14. Glover, F., Tabu search—part II, *ORSA Journal on Computing*, 2, 1, 4–32, 1990.
15. Gungor, A. and Gupta, S. M., A systematic solution approach to the disassembly line balancing problem, *Proceedings of the 25th International Conference on Computers and Industrial Engineering*, New Orleans, Louisiana, March 29–April 1, pp. 70–73, 1999.
16. Gungor, A. and Gupta, S. M., Disassembly line balancing, *Proceedings of the 1999 Annual Meeting of the Northeast Decision Sciences Institute*, Newport, Rhode Island, March 24–26, pp. 193–195, 1999.

17. Gungor, A. and Gupta, S. M., Issues in environmentally conscious manufacturing and product recovery: a survey, *Computers and Industrial Engineering*, 36, 4, 811–853, 1999.
18. Gungor, A. and Gupta, S. M., A solution approach to the disassembly line problem in the presence of task failures, *International Journal of Production Research*, 39, 7, 1427–1467, 2001.
19. Gungor, A. and Gupta, S. M., Disassembly line in product recovery, *International Journal of Production Research,* 40, 11, 2569–2589, 2002.
20. Gupta, S. M. and Taleb, K. N., Scheduling disassembly, *International Journal of Production Research*, 32, 1857–1866, 1994.
21. Gutjahr, A. L. and Nemhauser, G. L., An algorithm for the line balancing problem, *Management Science*, 11, 2, 308–315, 1964.
22. Hackman, S. T., Magazine, M. J. and Wee, T. S., Fast, effective algorithms for simple assembly line balancing problems, *Operations Research*, 37, 6, 916–924, 1989.
23. Hopgood, A. A., *Knowledge-Based Systems for Engineers and Scientists*, CRC Press, Boca Raton, FL, 1993.
24. Hu, T. C. and Shing, M. T., *Combinatorial Algorithms*, Dover Publications, Inc., Mineola, NY, 2002.
25. Kongar, E. and Gupta, S. M., A genetic algorithm for disassembly process planning, *Proceedings of the 2001 SPIE International Conference on Environmentally Conscious Manufacturing II*, Newton, MA, October 28–29, pp. 54–62, 2001.
26. Koza, J. R., *Genetic Programming: On the Programming of Computers by the Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.
27. Lambert, A. J. D., Disassembly sequencing: a survey, *International Journal of Production Research*, 41, 16, 3721–3759, 2003.
28. Lambert, A. J. D. and Gupta, S. M., *Disassembly Modeling for Assembly, Maintenance, Reuse, and Recycling*, CRC Press, Boca Raton, FL, 2005.
29. Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. and Shmoys, D. B., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley & Sons, New York, NY, 1985.
30. McGovern, S. M. and Gupta, S. M., Greedy algorithm for disassembly line scheduling, *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics*, Washington, D.C., October 5–8, pp. 1737–1744, 2003.
31. McGovern, S. M. and Gupta, S. M., Demanufacturing strategy based upon metaheuristics, *Proceedings of the 2004 Industrial Engineering Research Conference*, Houston, Texas, May 15–19, CD-ROM, 2004.
32. McGovern, S. M. and Gupta, S. M., Multi-criteria ant system and genetic algorithm for end-of-life decision making, *Proceedings of the 35th Annual Meeting of the Decision Sciences Institute*, Boston, MA, November 20–23, pp. 6371–6376, 2004.
33. McGovern, S. M. and Gupta, S. M., Local search heuristics and greedy algorithm for balancing the disassembly line, *The International Journal of Operations and Quantitative Management*, 11, 2, 91–114, 2005.
34. McGovern, S. M. and Gupta, S. M., Uninformed and probabilistic distributed agent combinatorial searches for the unary NP-complete disassembly line balancing problem, *Proceedings of the SPIE International Conference on Environmentally Conscious Manufacturing V,* Boston, MA, October 23–26, pp. 81–92, 2005.
35. McGovern, S. M. and Gupta, S. M., Ant colony optimization for disassembly sequencing with multiple objectives, *The International Journal of Advanced Manufacturing Technology*, 30, 5–6, 481–496, 2006.

36. McGovern, S. M. and Gupta, S. M., Computational complexity of a reverse manufacturing line, *Proceedings of the 2006 SPIE International Conference on Environmentally Conscious Manufacturing VI*, Boston, MA, October 1–4, CD-ROM, 2006.
37. McGovern, S. M. and Gupta, S. M., Deterministic hybrid and stochastic combinatorial optimization treatments of an electronic product disassembly line, Chapter 13, in Kenneth, D., Lawrence and Ronald, K., Klimberg (Eds.), *Applications of Management Science, Volume 12—Applications of Management Science: In Productivity, Finance, and Operations*, Elsevier Science, New-Holland, Amsterdam, pp. 175–197, 2006.
38. McGovern, S. M. and Gupta, S. M., Performance metrics for end-of-life product processing, *Proceedings of the 17th Annual Conference of the Production and Operations Management Society*, Boston, MA, April 28–May 1, CD-ROM, 2006.
AQ10 39. McGovern, S. M. and Gupta, S. M., A balancing method and genetic algorithm for disassembly line balancing, *European Journal of Operational Research*, Elsevier Science Publishers, in press.
40. McGovern, S. M., Gupta, S. M. and Kamarthi, S. V., Solving disassembly sequence planning problems using combinatorial optimization, *Proceedings of the 2003 Northeast Decision Sciences Institute Conference*, Providence, RI, March 27–29, pp. 178–180, 2003.
41. McGovern, S. M., Gupta, S. M. and Nakashima, K., Multi-criteria optimization for non-linear end of lifecycle models, *Proceedings of the Sixth Conference on Eco-Balance*, Tsukuba, Japan, October 25–27, pp. 201–204, 2004.
42. McMullen, P. R. and Tarasewich, P., Using ant techniques to solve the assembly line balancing problem, *IIE Transactions*, 35, 605–617, 2003.
43. Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, Mineola, NY, 1998.
44. Pinedo, M., *Scheduling Theory, Algorithms and Systems*, Prentice Hall, Upper Saddle River, NJ, 2002.
45. Ponnambalam, S. G., Aravindan, P. and Naidu, G. M., A comparative evaluation of assembly line balancing heuristics, *The International Journal of Advanced Manufacturing Technology*, 15, 577–586, 1999.
46. Reingold, E. M., Nievergeld, J. and Deo, N., *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1977.
47. Scholl, A., Balancing and sequencing of assembly lines, *Physica*, Darmstadt, 1995. AQ11
48. Suresh, G., Vinod, V. V. and Sahu, S., A genetic algorithm for assembly line balancing, *Production Planning and Control*, 7, 1, 38–46, 1996.
49. Torres, F., Gil, P., Puente, S. T., Pomares, J. and Aracil, R., Automatic PC disassembly for component recovery, *International Journal of Advanced Manufacturing Technology*, 23, 1–2, 39–46, 2004.
50. Tovey, C. A., Tutorial on Computational Complexity, *Interfaces*, 32, 3, 30–61, 2002.
51. Zeid, I., Gupta, S. M. and Bardasz, T., A case-based reasoning approach to planning for disassembly, *Journal of Intelligent Manufacturing*, 8, 97–106, 1997.

QUERY FORM

Taylor and Francis

Environment Conscious Manufacturing

JOURNAL TITLE:	**DK552X**
ARTICLE NO:	**Ch006**

Queries and / or remarks

Query No.	Details Required	Author's Response
AQ1	Are “CRT,” “RAM” well-known acronyms? If not please expand at first occurrence.	
AQ2	The acronym “EWP” has been used for both “early leaving workpieces” and “exploding workpieces.” Please check and provide two separate acronyms.	
AQ3	The text description for Figures 6.1 and 6.2 do not match the artwork provided for Figure 6.1 and 6.2, respectively. Please verify.	
AQ4	Is the change in the sentence “The other objectives…vertex selection.”OK?	
AQ5	“bin-backing problem” has been changed to “bin-packing problem.” Please confirm.	
AQ6	The sentence “Any of these parts…removal directions)” looks incomplete. Please check.	
AQ7	The word “increments” has been replace by “increases” here. Please check.	
AQ8	Is the change in the sentence “The DLBP *a priori*…with problem size.” OK?	
AQ9	Refs. 13 and 14 are not cited within text. Please cite them at appropriate places within text.	
AQ10	Please provide the publication details for Ref. 39, if available.	
AQ11	Please provide the volume number and the page range for Ref. 47.	