Journal of
INDUSTRIAL
TECHNOLOGY

Volume 17, Number 4 - August 2001 to October 2001

The Application of Multi-Level Genetic Algorithms in Assembly Planning

By Dr. Shana Shiang-Fong Smith (Shiang-Fong Chen) and Mr. Yong-Jin Liu

KEYWORD SEARCH

CAD
CAM
CIM
Manufacturing
NC/CNC

Reviewed Article

The Official Electronic Publication of the National Association of Industrial Technology • www.nait.org

The Application of Multi-Level Genetic Algorithms in Assembly Planning

By Shana Shiang-Fong Smith ***(Shiang-Fong Chen) and Mr. Yong-Jin Liu***

Shana Shiang-Fong Smith (Shiang-Fong Chen) is an assistant professor in the Department of Industrial Education and Technology at Iowa State University. Prior to joining Iowa State University, Dr. Smith worked at the Hong Kong University of Science and Technology and Tri-State University. She received the Ray Witt Gift from the Foundry Educational Foundation in 2000. Dr. Smith's research focuses on assembly/disassembly analysis, CAD/CAM, product design, virtual reality, and robot motion planning.

Yong-Jin Liu is currently a PhD candidate at Hong Knog University of Science and Technology. He received his B.Eng (1998) in Mechano-Electronic Engineering, Tianjin University, PRC, and Mphil (1999) in Mechanical Engineering, Hong Knog University of Science and Technology. His research interests include geometric modeling, visualization, reverse engineering, assembly sequence planning and computer graphics.

Introduction

In the 1950s, the market for U.S products was eight times larger than the next largest (Preston 1993). However, the U.S. held tightly to traditional mass production techniques, while, beginning in the 1950's, Japan and then Germany embraced new manufacturing techniques: concurrent engineering, manufacturing automation, and flexible computer-integrated manufacturing. As a result, from 1960 – 1989, U.S. share of world trade in manufactured goods fell from 20% to 17%, from 1st place to 3rd place, behind Japan and Germany. New manufacturing techniques brought customers more customized products, with new models in shorter periods of time. Now, customers expect progressively higher-quality products, at lower cost, more quickly, and more customized to individual tastes.

In the new, more highly automated manufacturing environment, one technology has been identified as critical to all manufacturing areas: assembly of parts to produce a final product (Dhar 1989, Preston 1993). For many products, assembly activities account for a major part of product production time and, thus, production cost. Therefore, industrialized countries, such as Japan and Germany, that have adopted flexible automated assembly techniques, have met customer product demands, reduced costs, and captured growing world-wide market share (Dhar 1989, Preston 1993).

Traditionally, experienced assembly technicians have scheduled a product assembly plan. However, since computers and robots have been replacing assembly workers, automatic assembly planning has become more important in automated manufacturing systems. In addition, in modern computer-integrated manufacturing systems, assembly plans may need to be calculated, or re-calculated, quickly to meet changing manufacturing system conditions.

Automatic assembly sequence planning (ASP) plays a vital role in planning manufacturing processes for products during product development and, thus, plays an important role in computer-aided intelligent concurrent engineering. For example, in an automotive factory, it might take hours for each refixturing or reorientation. Therefore, proper product design for assembly and manufacturing, as well as choice of an efficient assembly plan, greatly determines lead-time, production cost, and, thus, potential product success. Automated assembly planning helps provide tools needed for thorough evaluation of potential assembly plans for any given product.

Valid assembly plans must satisfy all geometrical constraints between components in a product, so that parts will not collide during product assembly. In addition, an efficient assembly plan should be optimized for goals related to physical constraints; e.g., minimized for the number of assembly reorientations required, minimized for the number of tool changes required, and minimized for the number of fixtures required. Assembly plans that have not been optimized with respect to physical constraints may result in difficulties when assembling the product.

ASP is a complex combinatorial optimization problem (De Fazio and Whitney 1987, Baldwin *et al.* 1991, Delchambre 1992, Laperriere and Elmaraghy 1996, Gottipolu and Ghosh 1997). Prior related research reports two primary categories of solution

algorithms, graph-theory based algorithms and artificial intelligence based algorithms.

Most reported algorithms solve assembly problems by graph searching. In most graph searching algorithms, each connection between two components, or two subassemblies, is called a *liaison*. First, a liaison diagram is built, in which vertices represent components and edges represent mating relations between components (see Figure 1). Next, all possible subassemblies are generated from the liaison graph by decomposing the graph, subject to geometric constraints. Then, possible assembly sequences are evaluated, based on given physical constraints, to determine the most suitable sequence. Such exhaustive searching methods require substantial computational resources even for a simple structure.

More recent research has begun to apply artificial intelligence based algorithms to solve the assembly sequence planning problem. In particular, motivated by the success of genetic algorithms (GAs) in solving difficult and complex combinatorial problems (Gen and Cheng 1997), GAs have been applied to assembly sequence planning to find optimal or near-optimal assembly plans for a structure. In contrast to graph searching techniques, GAs do not generate, and then evaluate, all possible candidate solutions when searching for an optimized assembly plan. Rather, GAs search for an optimized assembly plan using a directed stochastic search of the product's solution space of possible assembly plans. Thus, GAs attempt to find optimized assembly plans, while analyzing only a small number of possible solutions. The goal, then, of using GAs for assembly planning is to dramatically reduce time required to find an acceptably optimized assembly plan, for any given product structure. However, while graph theory based algorithms can find absolute optimal solutions, if they exist, GA based algorithms only optimize (or improve) with respect to an initially supplied set of assembly plans, for a given set of goals.

GAs use a stochastic search algorithm that attempts to mimic natural biological processes that result in genetic variation from generation-to-generation in living species. In conventional GAs, a candidate solution is typically encoded as a bit string, called a *chromosome*. For the example in Figure 1(a), an assembly plan, encoded as a chromosome, might be the sequence A-F-E-D-C-B, representing the assembly order for parts in the structure. An initial population of solutions is first created (either manually, by a technician, or automatically by a computer program) and the fitness of each individual in the population is evaluated. GAs usually evaluate an individual's fitness, with respect to the entire population, by applying a function that incorporates, for example, measures of assembly reorientations required, assembly tool changes required, and/or assembly fixtures required. Next, GAs select individuals, or pairs of individuals, based on fitness, from which to produce a next generation of individual assembly plans.

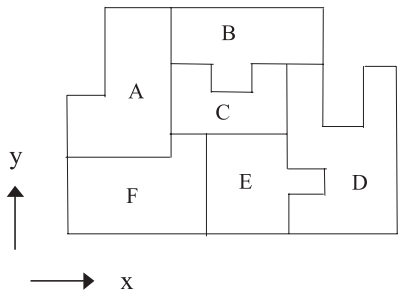
For producing any individual offspring assembly sequence, the GA chooses one genetic operator, in our case (*crossover*, *mutation*, or *cut-and-paste*), based upon pre-defined probability settings. For example, before running a GA, the probability settings for (*crossover*, *mutation* or *cut-and-paste*) might be set to (40%, 20%, 40%). We define a set of probability settings for a given set of genetic operators as a *genetic operator probability setting* (GOPS). The GA then applies the chosen operator to chromosome string(s) selected from the population, based on relative fitness, to produce the next-generation population.

A crossover operator exchanges part assembly order information between two chromosomes. A mutation operator exchanges the assembly order of two parts within a single selected chromosome. A cut-and-paste operator cuts a group of parts from one location in a chromosome and pastes the group of parts to a new location in the chromosome. Later sections describe genetic operator functions in more detail.

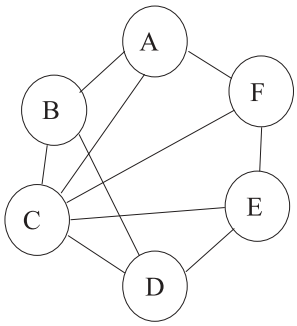
Most GAs for assembly planning use fixed population sizes from generation to generation, to limit algorithm run time. Thus, when the GA generates enough new valid offspring to fill the next generation (valid with respect to geometric constraints), the old population is replaced by the new offspring population and a fitness value for each new member is then calculated. The GA process of generating new higher-fitness offspring from the previous generation repeats until pre-defined termination criteria are satisfied. Common termination criteria used include a run-time limit, the number of generations, or a pre-defined desired assembly sequence fitness level.

Bonneville *et al*. (1995) first introduced, and demonstrated the feasibility of using, a genetic algorithm for assembly planning. However, Chen (1998) noted that GA-based assembly planning can result in inconsistent output performance; GA-based assembly planning is heavily affected by the initial set of valid assembly plans (initial population) supplied by an expert. Smith

Figure1: A structure and its liaison graph



(a)



(b)

and Chen (2000) also demonstrated a technique for automated initial population generation, to provide more consistent, and improved overall performance. Bonneville *et al.*'s early GA-based assembly planner used two basic genetic operators, *crossover* and *mutation*. However, Goldberg (1989) showed that genetic algorithms that use only crossover and mutation operators are limited in performance. Sebaaly and Fujimoto (1996) also proposed a genetic algorithm for solving assembly sequence planning problems. To provide a more thorough search of a product's assembly plan solution space and, thus, more consistent output results, Sebaaly and Fujimoto proposed an algorithm for clustering the search space into families of similar sequences, where each family contains only one feasible sequence. Subsequently, the best solutions are found without searching the complete solution space. Sebaaly and Fujimoto do not clearly describe their clustering method, however, clustering appears to be a complex and time-intensive operation. Chen (1998) proposed a simplified genetic algorithm for automatically generating assembly sequences. Chen used five genetic operators (crossover, mutation, cut-and-paste, break-and-joint, and reproduction) to promote a more thorough search of the product's assembly plan solution space.

All early GA-based assembly planners use fixed, user-selectable genetic operator probability setting (GOPS). However, Chen (1998) demonstrated that different GOPS leads to different final result performance. In addition, Kim *et al.* (1997) pointed out that high mutation rates reduce premature convergence and improve overall final-solution quality. Chen also noted that it is difficult to manually determine the best GOPS for different assembly planning problems. Thus, an automated technique for choosing GOPS to further improve overall GA-based assembly planner performance is critical.

Purpose

The genetic assembly planners discussed above all use classical single-level genetic algorithms, with GOPS that remain fixed throughout the entire algorithm process. As noted, Kim *et al.* (1997) pointed out that high mutation probability settings reduce premature convergence and improve overall final-solution quality. On the other hand, higher mutation probabilities reduce overall algorithm convergence rate, leading to longer run times, or may actually result in GA non-convergence. As a result, rather than using a single-level GA with fixed GOPS, in this paper, we dynamically update GOPS, to optimize the effectiveness of each type of genetic operator during the solution searching process. Thus, we propose a further improvement in state-of-the-art GA ASP; we propose using a multi-level genetic assembly sequence planner that dynamically optimizes GOPS selection, to improve solution quality and to speed up the searching process for an ASP solution.

Overview of Our Multi-level Genetic Assembly Planner

Our multi-level genetic assembly planner uses two genetic algorithms, working at two different levels. A low-level (level-1) GA generates new assembly sequence populations, based on a current GOPS. A high-level (level-2) GA optimizes and updates GOPS for the low-level GA. Figure 2 illustrates our multi-level genetic assembly planner, in block diagram form.

In our planner, low-level (level-1) assembly sequence selection is dynamically optimized by our high-level (level-2) GA GOPS selection . We use a dynamic control mechanism for generating high-level GA offspring. High-level GA offspring represent new GOPS used to synchronously update (and, thus, enhance performance of) our low-level GA.

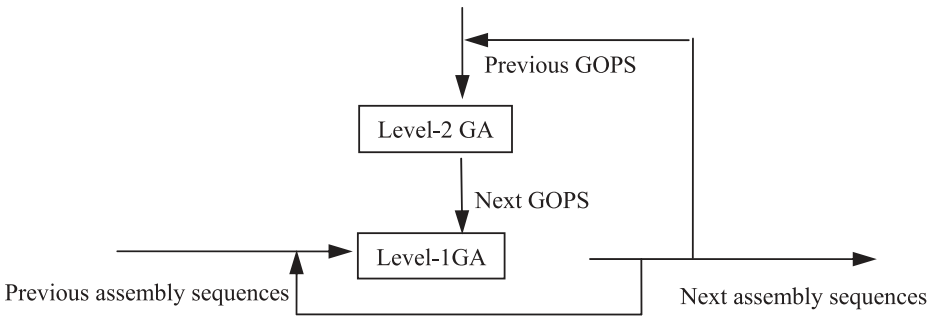
In conventional "GA within GA" approaches, each high-level GA generation incorporates a full low-level GA run. Thus, conventional hierarchical GAs are very time-consuming. Unlike conventional "GA within GA" approaches (used in research areas other than ASP), during our level-1 GA execution, our level-2 GA synchronously updates GOPS during, rather than only after, each full level-1 GA run. Thus, our algorithm dramatically reduces searching time over conventional "GA within GA" approaches. Experimental results show that our multi-level genetic assembly sequence planner solves combinatorial ASP problems more quickly and reliably than prior classical single-level GA approaches.

Level-1 Genetic Algorithm

Level-1 GA Encoding – Chromosome Representation for Assembly Plans

The first step in applying GAs requires encoding possible solutions into chromosomes. In this paper, the chromosomes in our level-1 GA represent assembly sequences (assembly plans). Since an assembly plan is usually represented by an assembly tree, our solutions are encoded as trees. For example, Figure 3 shows some example assembly trees that represent valid or invalid assembly sequences for the structure in Figure 1(a).

Figure 2: Overview of the proposed multi-level GA assembly planner



Level-1 GA Fitness Function

As a second step in creating our level-1 GA, we must define a fitness function for evaluating chromosome (or assembly sequence) quality. Suppose our goal is to find a sequence with minimum reorientations. We define the fitness function for a valid assembly plan Π by

$$original\_fitness = n(\Pi) \quad (1)$$

where $n(\Pi)$ is the number of reorientations required for sequence Π. We can calculate the number of reorientations and the validity for any sequence Π using methods proposed by Chen (1998).

In order to help our searching process converge within a reasonable number of generations, we use a modified linear scaling scheme, to adjust fitness values:

$$new\_fitness = \begin{cases} (f_a / u_{avg})u, & u \le u_{avg} \\ f_a + \dfrac{f_b - f_a}{u_{\max} - u_{avg}}(u - u_{avg}), & u > u_{avg} \end{cases} \quad (2)$$

where u represents original assembly sequence fitness value calculated from Eq. (1), u_{avg} is the average original fitness value for the current population, and u_{max} is the maximum original fitness value for the current population. By definition $f_b > f_a$, and we can select f_a and f_b for the problem at hand.

Level-1 GA Genetic Operators

Given a proper assembly model and fitness function, we next need to define a set of genetic operators for generating diverse offspring chromosomes from any given assembly sequence population. For our current study, we use three genetic operators: *crossover, mutation,* and *cut-and-paste*.

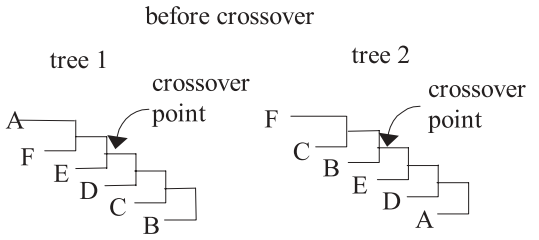
A tree crossover operator is applied to swap two sub-trees between two assembly trees. The crossover point is chosen randomly. Upper sub-trees remain the same. Nodes in the lower sub-trees are reordered to follow the sequence in the other assembly tree. One example is shown in Figure 3 (a). The sequence of parts A-F-E in the upper sub-tree of tree 1 is unchanged for tree 1¢. However, remaining parts D-C-B follow their precedence in tree 2, so their new order in tree 1¢ becomes C-B-D.

A tree mutation operator is used to swap two nodes within a single assembly tree. The two swapped nodes are chosen randomly (see Figure 3 (b)). A cut-and-paste operator is used to promote assembly sequence solutions that contain sub-trees that meet certain criteria (see Figure 3 (c)). A cut node and a paste position are chosen randomly. If the nodes adjacent to the cut node satisfy the same criterion, e.g., same tool used or same assembly direction, they are cut and then pasted to the new paste position together.

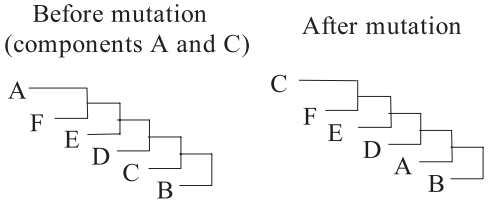
Level-1 GA Searching Process

Figure 4 illustrates our level-1 GA searching process, as a flow diagram. To generate offspring with high fitness values, a good genetic operator selection mechanism is necessary. In our algorithm, the genetic operator selection mechanism is based upon the given GOPS, provided by our level-2 GA. Our level-1 GA stops when population fitness stabilizes.

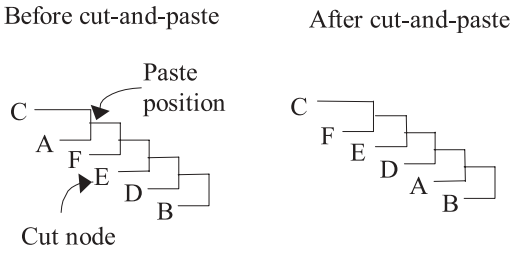
Figure 3: Genetic operators in assembly trees



(a) crossover operator



(b) mutation operator



(c) cut-and-paste operator

Level-2 Genetic Algorithm

Conventional genetic algorithms use static GOPS for genetic operator selection. However, for improved genetic searching, GOPS should be different in different search stages (Kim *et al.* 1997). As a result, we propose a level-2 GA for dynamically optimizing GOPS, at each level-1 GA searching stage.

Level-2 GA Encoding – Chromosome Representation for GOPS

Our level-2 GA population is composed of candidate level-1 GOPS. The chromosomes in our level-2 GA are level-1 GOPS vectors. For example, a GOPS vector

$$V_1 = \{C_1, M_1, CP_1\} = \{40.0, 40.0, 20.0\}$$

represents a level-1 GA crossover probability setting, mutation probability setting, and cut-and-paste probability setting of 40.0%, 40.0%, and 20.0%, respectively. In other words, when selecting the level-1 genetic operator to apply for generating a next generation of assembly sequences, crossover would be selected with a 40% probability, mutation with a 40% probability, and cut-and-paste with a 20% probability.

Level-2 GA Fitness Function

Each vector in our current level-2 population represents a candidate level-1 GOPS. For each candidate GOPS in level-2, our level-1 GA tends to generate a corresponding local-optimal assembly sequence plan. We define the resulting assembly sequence fitness value in our level-1 GA as the fitness of a corresponding GOPS vector in our level-2 GA.

Level-2 Genetic Operators

We propose a *hybrid crossover* operator and *a hybrid mutation* operator for our level-2 GA. For our hybrid crossover operator, given two parents

$Parent_1 = \{C_1, M_1, CP_1\}$

$Parent_2 = \{C_2, M_2, CP_2\}$

we define

$$C_{new} = C_1 \quad rand \quad (C_1 \quad C_2)$$

$$M_{new} = M_1 \quad rand \quad (M_1 \quad M_2)$$

$$CP_{new} = CP_1 \quad rand \quad (CP_1 \quad CP_2)$$

where *rand* is a random value between 0 and 1. We then define an offspring from $Parent_1$ and $Parent_2$

$$Offspring = \{C_{new}, M_{new}, CP_{new}\}$$

For our hybrid mutation operator, we select a parent GOPS and randomly generate a GOPS chromosome as follows.

$$Parent = (C_p, M_p, CP_p)$$

$$Random = (C_r, M_r, CP_r)$$

We then hybridize our selected parent and random chromosomes:

$$C_{new} = C_p \quad rand \quad (C_p \quad C_r)$$

$$M_{new} = M_p \quad rand \quad (M_p \quad M_r)$$

$$CP_{new} = CP_p \quad rand \quad (CP_p \quad CP_r)$$

Finally, we define an offspring GOPS chromosome

$$Offspring = (C_{new}, M_{new}, CP_{new})$$

For our level-2 GA, our hybrid crossover operator selection probability is set to 90% and our hybrid mutation operator selection probability is set to 10%.

Observation

Applying our ASP algorithm to many test cases, we have found that, for a product with *n* components, our level-1 GA population undergoes most significant variation in an early searching stage, that ends after approximately *n* generations, and reaches a stable state after approximately *5n* generations. Thus, the final result of a full-run level-1 GA can be approximated by the result at the end of early-stage searching. Figure 5 presents one example that shows our level-1 GA running performance for the 19- component assembly in Figure 6.

To implement our synchronous multi-level GA, we need on-line runtime information exchange between our level-1 GA and our level-2 GA. Thus, for each level-2 generation, we run our underlying level-1 GA for a limited number of iterations, *n* genera-

Figure 4: Level-1 GA

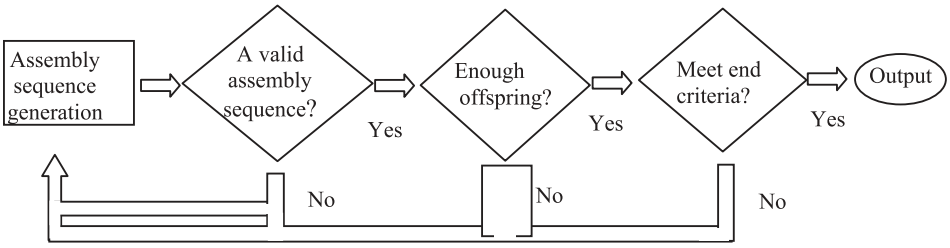
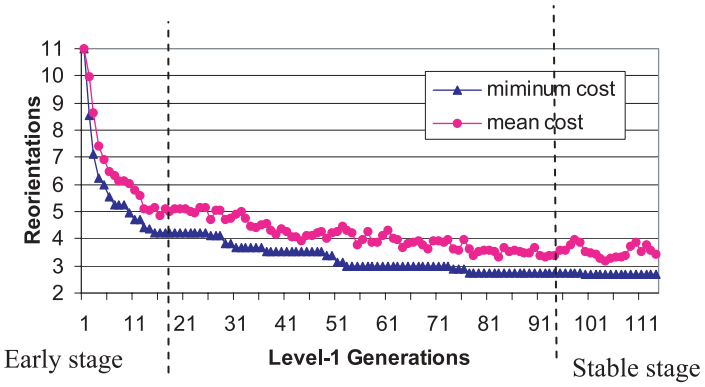


Figure 5: Running performance of a full-run level-1 GA for a 19-component structure



tions. We then feed back current level-1 GA population fitness to our level-2 GA, for creating a new level-2 GA generation and for then updating our level-1 GA GOPS.

Synchronous Multi-level Searching

The key feature of our proposed hierarchical ASP GA algorithm is the dynamic cooperation between our two associated GAs. From our prior observation, in our two-level GA, we only need to measure partial low-level GA performance rather than performance over a full low-level GA run. Thus, our algorithm dramatically reduces required searching time.

Implementation

Our multi-level genetic assembly planner has been fully implemented in C++ on a 450 MHz Pentium II PC with 256 MB memory. First, we applied our planner to the 19-component assembly shown in Figure 6. Our multi-level genetic assembly planner finds a 2.03-reorientation solution in 23 level-2 generations (see Figure 7). For comparison with our multi-level GA, we tested two sets of fixed GOPS (given in Table 1) for our example in Figure 6. All results are averaged over 30 different runs. From our comparison, we can see that using fixed GOPS setting 1 leads to a 2.867-reorientation assembly sequence, and using fixed GOPS setting 2 leads to a 2.70-reorientation assembly sequence.

We further confirm performance of our multi-level genetic assembly sequence planner for an 11-component assembly (see Figure 8) and a complex 48-component assembly (see Figure 10). Results for our 11-component example and our 48-component example are presented in Figure 9 and Figure 11, respectively. Run-times for our three example products are shown in Table 2. We find that run-time depends primarily on the number of product components and on product assembly complexity. Our experimental results show that our multi-level genetic assembly sequence planner both improves solution quality and speeds up the searching process for an optimized assembly sequence.

Figure 6: A 19-component assembly drawing.

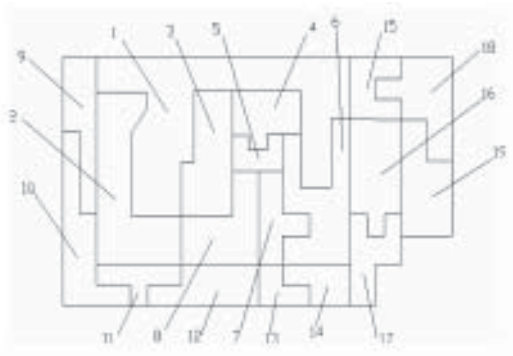


Figure 7: Results using our multi-level genetic planner for our Figure-6 example

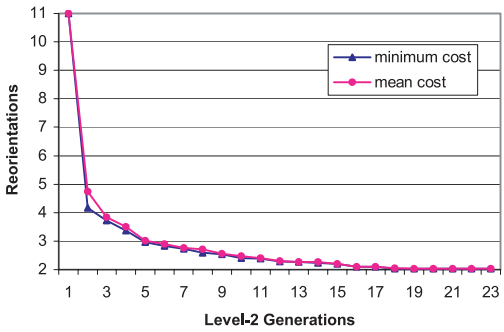


Figure 8: An 11-component product

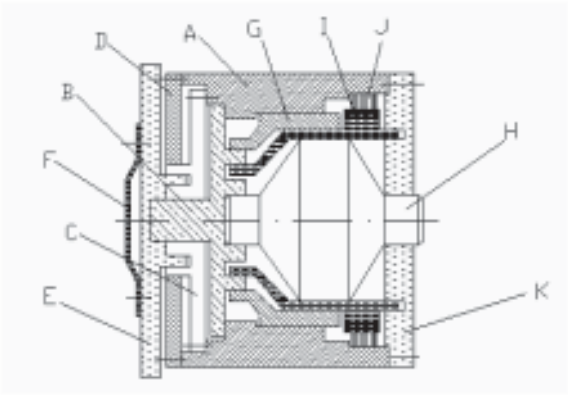


Table 1: Comparisons between fixed GOPS settings and proposed dynamic GOPS

Fixed GOPS			Dynamic GOPS
	Setting 1	Setting 2	Proposed Multi-Level Genetic Algorithm
Crossover	40%	10%	
Mutation	20%	80%	
Cut-and Paste	40%	10%	
Minimum number of reorientations	2.867	2.70	2.03

Conclusions

History has taught us an important lesson: manufacturing is a prime generator of wealth and is critical in establishing a sound basis for economic growth (Comerford 1993). Customer demand now requires more flexible automated manufacturing systems. Assembly sequence affects manufacturing equipment layout and manufacturing process selection. Given feasible assembly sequences, we can then determine optimal machine selection and equipment layout for each sequence. On the other hand, feasible manufacturing options determine feasible assembly sequences. Therefore, automated assembly sequence planning has become more important in automated manufacturing systems.

In this paper, we propose an efficient multi-level genetic assembly sequence planner. In our multi-level assembly sequence planner, a level-1 GA is used to generate an optimal assembly sequence based on optimized GOPS generated by a level-2 GA. By using dynamic GOPS, our genetic planner efficiently generates global-optimal or near-global-optimal assembly sequences within a small number of GA generations. Our planner not only considers geometric constraints, but also optimizes for physical constraints. Our test results show that our multi-level genetic assembly sequence planner is both fast and efficient at finding optimized assembly sequences for a given product structure.

Since the multi-level genetic assembly sequence planner works well on all of the tested product structures, in the next phase of our project, we plan to cooperate with manufacturing companies to further test our multi-level genetic assembly sequence planner, to evaluate the impact of the automated planner on their manufacturing processes, and to quantify monetary and personnel impacts.

References

Baldwin, D., Abell, T., Lui, M., De Fazio, T. and Whitney, D. (1991). An integrated computer aid for generating and evaluating assembly se-

Figure 9: Results using our multi-level genetic planner for our Figure-8 example

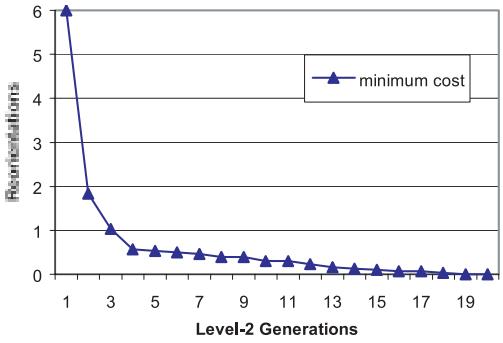


Figure 10: A 48-component headstock product

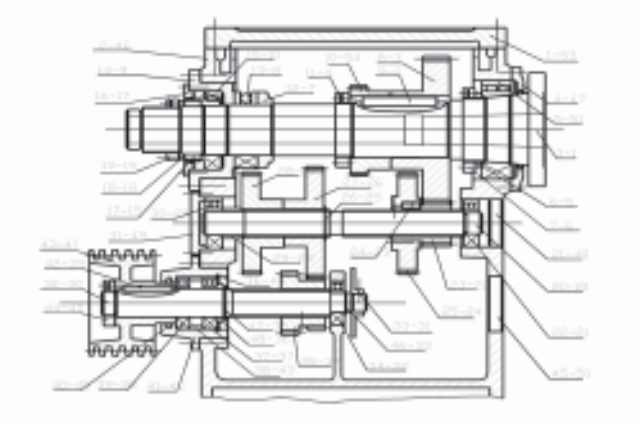


Figure 11: Results using our multi-level genetic planner for our Figure-10 product

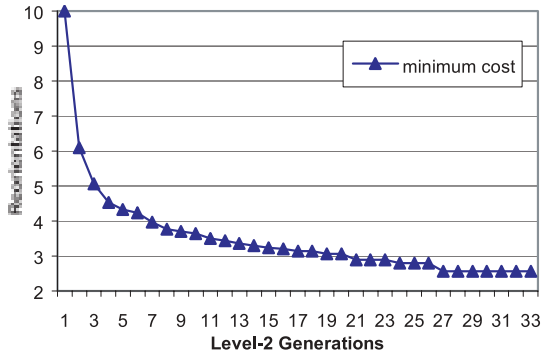


Table 2: GA run times for three example products (all results are averaged over 30 different runs)

	19-component (Figure 6)	11-component (Figure 8)	48-component (Figure 11)
CPU time (sec)	10	3	56

quences for mechanical products. IEEE Transactions on Robotics and Automation, vol. 7, no. 1, pp. 78-94.

Bonneville, F., Perrard, C., and Henrioud J, M. (1995). A genetic algorithm to generate and evaluate assembly plans. IEEE Symposium on Emerging Technology and Factory Automation, vol. 2, pp. 231-239.

Chen, S. F. (1998). Assembly planning-a genetic approach. Proceedings of the 24th ASME Design Automation Conference. September 12-16 (Atlanta, Georgia), paper no. DETC98/DAC-5798.

Comerford, R. (1993). A preview of the 21st century. IEEE Spectrum, v 30, n 9, September, pp. 82 – 85.

De Fazio, T. L. and Whitney, D. E. (1987). Simplified generation of all mechanical assembly sequences. IEEE J. Robotics and Automation, RA-3, vol. 6, pp. 640-658.

Delchambre, A. (1992). Computer-Aided Assembly Planning (Chapman & Hall, London, UK).

Dhar, U. R. (1989). Flexible manufacturing systems: major breakthrough in manufacturing management. Engineering Management International, vol. 5, no. 4, May, pp. 271 – 277.

Gen, M. and Cheng, R. W. (1997). Genetic Algorithms and Engineering Design (John Wiley & Sons, Inc).

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning (Addison-Wesley).

Gottipolu, R. B. and Ghosh, K. (1997). Representation and selection of assembly sequences in computer-aided assembly process planning. International Journal of Product Research, 35(12), pp. 3447-3465.

Kim, B. M., Kim, Y. B., and Oh, C. H. (1997). A study on the convergence of genetic algorithms. Computer Industrial Engineering, 33(3-4), pp.581-588.

Laperriere, L. and Elmaraghy, H. (1996). GAPP: A generative assembly process planner. Journal of Manufacturing Systems, 15 (4), pp.282-293.

Preston, M. E. (1993). Mechatronic product development survey and examples. Mechatronics, edited by Hewit, J. R. (Springer-Verlag, Wien-New York).

Sebaaly, M. F. and Fujimoto, H. (1996). A genetic planner for assembly automation. Proceedings of the IEEE Conference on Evolutionary Computation, pp. 401-406.

Smith, G. C. and Chen, S. F. (2000). Automated Initial-Population Generation for Genetic-Algorithm-Based Assembly Planning. 7th ASME Flexible Assembly Conference. September 10-13, 2000 (Baltimore, Maryland), paper number DETC2000/FLEX-14464