Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 53 (2016) 44 – 49

www.elsevier.com/locate/procedia

The 10th International Conference on Axiomatic Design, ICAD 2016

An improved axiomatic design approach in distributed resource environment, part 2: Algorithm for functional unit chain set generation

Bin Chen[a,b,*], Jun Liu[a,b], Youbai Xie[a,b]

[a]*State Key Laboratory of Mechanical System and Vibration, Shanghai Jiao Tong University, Shanghai, 200240,China*
[b]*School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai, 200240,China*

* Corresponding author. Tel.: +86-15800789240; fax: +0-000-000-0000. *E-mail address:* chenbinsun@outlook.com

Abstract

To technologically achieve the transformation from functional requirements (FRs) to design parameters (DPs) in the axiomatic design (AD) theory, this paper employed the basic hypothesis, definitions and model developed in PART 1 of our study to establish an algorithm for functional unit chain set (FUCS) generation. This algorithm is established on two mappings from the developed conceptual foundation to the graph theory, i.e. treating functional units (FUs) as nodes, and treating connections between FUs as edges. With this two mappings, a graph consisting of FUs can be created, which is called the functional unit graph (FUG). Based on the depth-first search (DFS) strategy in the graph theory, the algorithm generating FUCSs from the FUG was finally established. As an additional function, a preliminary evaluation for the generated FUCSs was also proposed. To prove the feasibility of this algorithm, a solar-powered wiper blades, and the load module of a friction-abrasion testing machine were designed as illustrations.

Keywords: Generating algorithm, functional unit chain set, depth-first search, distributed resource environment

1. Introduction

As mentioned in the preceding paper PART 1, how to transform FRs into DPs plays a key role in the AD theory, and how to generate FUCSs from the distributed resource environment consisting of FUs plays a key role in achieving this transformation. However there is still no good technological method to solve this problem. Therefore, finding a similar circumstance with mature solution may be a feasible study direction.

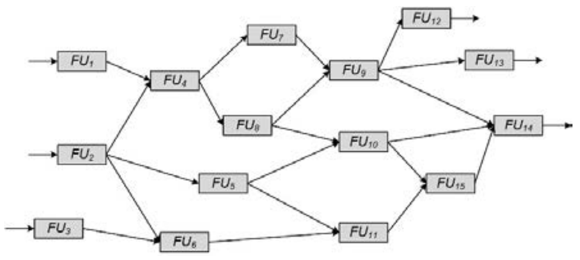
Some researchers used graph theory to describe and study distributed resources, which might be inspiring to solve this problem. Mei et al. studied the distributed containment control problem for networked Lagrangian systems with multiple dynamic leaders in the presence of parametric uncertainties under a directed graph that characterizes the interaction among the leaders and the followers. [1] Olfati-Saber and Murray proposed a graph rigidity and distributed formation stabilization of multi-vehicle systems. [2] Shen and Tsai proposed a graph matching approach for solving the task assignment problem encountered in distributed computing systems. [3] Gonzalez et al. developed the distributed graph-parallel computation on natural graphs. [4] Cheung proposed graph traversal techniques and the maximum flow problem in distributed computation. [5] Chen and Xie proposed a computer-assisted automatic conceptual design system for the distributed multi-disciplinary resource environment. [6] Although these studies focused on other different disciplinary domains, they can still prove the possibility to use graph theory describing and studying distributed resource environment consisting of FUs in our study.

If the FUs here are treated as nodes in graph theory, and meanwhile, the connections of FUs here are treated as edges in graph theory, the distributed resource environment can be transformed into a graph consisting of FUs, which connect to each other. So the problem, generating FUCS from the distributed resource environment consisting of knowledge services they providing different kinds of FUs via Internet geographically located in the world, is transformed into finding a path with given starting and ending points in a graph under given conditions, which has been already solved with mature technology. Kwan proposed this graph theory question with the description of Chinese postman problem. [7] After that, Benavent et al. studied this problem and found that with the algorithm proposed by Dijkstra [8] this problem can be solved. [9] Furthermore, Tarjan proposed a more efficient algorithm called Depth-First Search (DFS) to solve this problem successfully. [10] With the progress of computational technology, this DFS method can be successfully applied on many practical industrial domains with computers. Gregor and Lumsdaine developed a generic library for distributed graph computations. [11] Barooah and Hespanha proposed a graph effective resistance and distributed control. [12]

Inspired by these above researches, we proposed an algorithm for FUCS generation based on DFS. The FUCS generation is a necessary step in transforming FRs to DPs, and is a FRs knowledge integrating process before DPs knowledge integration, which aims in finding an ordered FUs chain integrated from FUs on the basis of knowledge of FUs to meet an indivisible FR in design. The whole work will be introduced in 4 parts, i.e. functional unit graph (FUG), algorithm for FUCS generation, preliminary evaluation for FUCSs, and the application and implementation. All these works will be concluded in the final of this paper.

2. Functional unit graph (FUG)

In the graph theory, a graph consists of many nodes connected to each other. The connections are defined as edges. This situation is suitable for the distributed resource environment consisting of FUs. If we treat FUs as nodes, treat their connections as edges, we can get a graph consists of FUs and their connections as shown in Figure. 1. This graph is a functional unit graph (FUG), and FUCSs are generated from



it.

Fig.1 An FUG with several kinds of connections as an illustrative example

3. Algorithm for FUCS generation

Based on DFS of graph theory, we proposed the algorithm to generate FUCS from an FUG. Before introducing the detail algorithm flow, some relevant assistant variables and the basic principle of the algorithm should be introduced first.

3.1. Relevant assistant variables and the basic principle of the algorithm

To generate an FUCS from the FUG, the first step is finding the FUs which can be the first one in the chain, and we call them head functional units (HFU). An HFU has the same input as the input of the objective F which has already been obtained by designers from the objective FR. After finding out all the HFUs, we pick one of them, and search its successors based on the edges. If one of its successors has the same output as the output of the objective F, we find out an objective FUCS. If none of the successors has the same output as the output of the objective F, we pick one of the successors, and keep on searching its successors, and repeat the above process, until we find out all the objective FUCSs. During this process, we set a maximum of the length of the objective FUCSs, if the length of the generating FUCS exceeds the maximum, we give up generating it.

Now, we introduce the following relevant assistant variables to achieve the above algorithm principle.

- Head functional unit tag.

  It is a logic variable attached to the FU, if the FU is an HFU, its value is 1, and if not, its value is 0. So, every FU has an HFU tag, and the value of this tag is helpful for the algorithm to determine if this FU is an HFU. Once this FU is determined as an HFU, the generating process of a new FUCS will begin from this FU.

- Stack.

  It is used to store the generating FUCS. Its bottom is an HFU. If its top FU has the same output as the output of the objective F, the FUs stored by the stack construct an objective FUCS, from the bottom to the top. The FUs can only be deleted and added to the stack from the top, which supports the searching process of FUs.

- Functional unit pointer.

  This pointer points to the FU which is being tested.

- Edge visiting tag.

  It is used to record the visiting situation of an edge. If this edge is visited, the value is 1, and if not, the value is 0.

- Functional unit visiting tag.

  It is used to record the visiting situation of an FU. If this FU is visited, the value is 1, and if not, the value is 0.

- The maximum of the length of functional unit chain set.

  It is represented as l_{max} and is used to control the length of the generating FUCS.

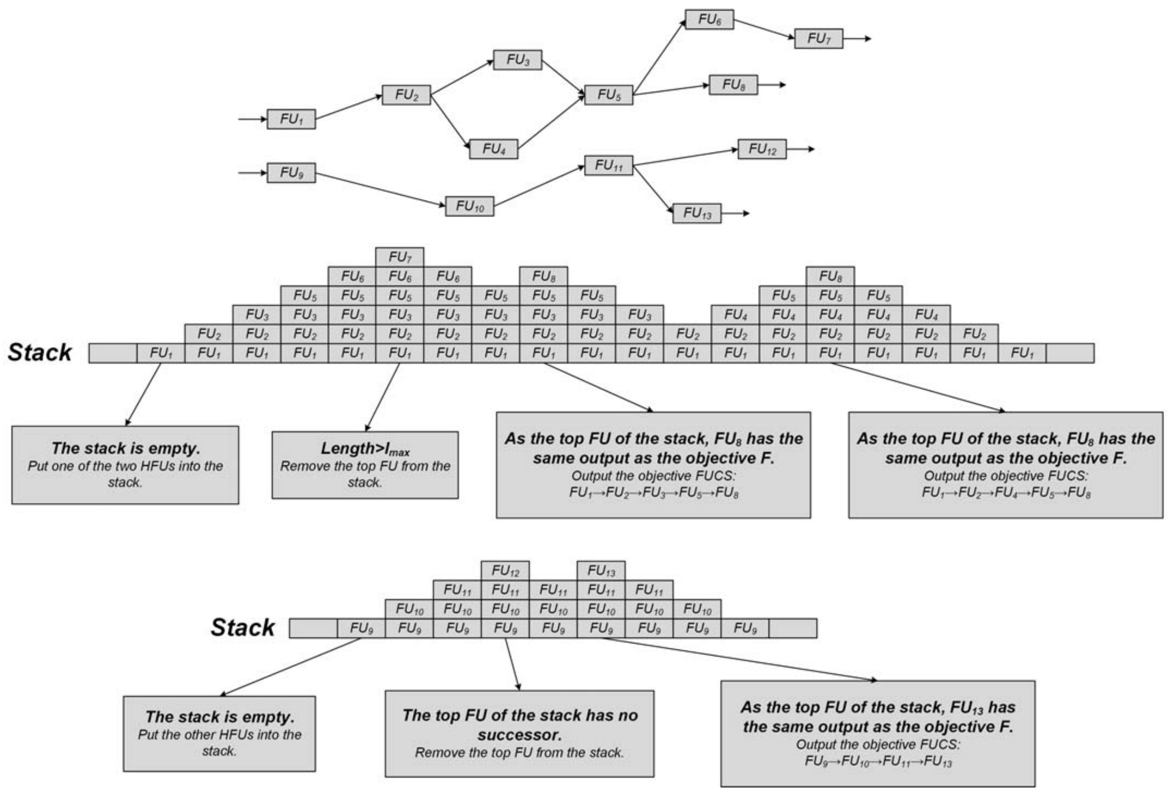

Stack
The stack is empty.
Put one of the two HFUs into the stack.
Length>l_{max}
Remove the top FU from the stack.
As the top FU of the stack, FU_8 has the same output as the objective F.
Output the objective FUCS:
$FU_1 \rightarrow FU_2 \rightarrow FU_3 \rightarrow FU_5 \rightarrow FU_8$
As the top FU of the stack, FU_8 has the same output as the objective F.
Output the objective FUCS:
$FU_1 \rightarrow FU_2 \rightarrow FU_4 \rightarrow FU_5 \rightarrow FU_8$
The stack is empty.
Put the other HFUs into the stack.
The top FU of the stack has no successor.
Remove the top FU from the stack.
As the top FU of the stack, FU_{13} has the same output as the objective F.
Output the objective FUCS:
$FU_9 \rightarrow FU_{10} \rightarrow FU_{11} \rightarrow FU_{13}$

Fig.2 An operating example for illustrating the basic principle of the algorithm

The stack is the core viable, and it changes during the process of the algorithm. As shown in Figure. 2, in the FUG, there are 13 FUs, i.e. FU1-FU13. The algorithm will find objective FUCSs from this FUG. The changing process of the stack is shown in the figure. At the beginning, the stack is empty. There are only two HFUs, i.e. FU1 and FU9. So there may be two kinds of possible objective FUCSs which start from FU1 and FU9 respectively, and the algorithm should try to generate them respectively. Here, the algorithm will first try to generate the possible objective FUCSs starting from FU1. At this time, the bottom of the stack is always FU1. When the generation of these possible objective FUCSs is completed, the algorithm should empty the stack and push FU9 into it as the bottom, so that, the generation of the other kind of possible objective FUCSs can begin.

So, first pick FU1 into the stack. After that, along the corresponding edges, FU2, FU3, FU5, and FU6 are pushed into the stack one by one, but none of them has the same output as the output of the objective F. So, keep on push FU7 into the stack. At this time, the length of the generating FUCS exceeds the maximum, so FU7 is directly removed. And the top FU will be tested. Both FU8 and FU6 have no successor, so they are removed. As the new top, FU5 has a successor FU8, so push FU8 into the stack. The output of FU8 meets the objective F, so output the FUs of the stack from the bottom to the top as an objective FUCS. Keep on testing the top FU of the stack. FU3 and FU5 both have no unvisited successor, so they are removed. As the new top, FU2 has an unvisited successor FU4, so push FU4 into the stack. But the output of FU4 cannot meet the objective F, so put its successor FU5 into the stack. And as the same situation, FU8 is pushed into the stack. The output of FU8 meets the objective F, so output the FUs of the stack from the bottom to the top as another objective FUCS. After that, keep on testing the top FU of the stack. FU8, FU5, FU4, FU2, and FU1 all have no unvisited successor, so they are all removed one by one.

At this time, the stack is empty again, push the other head functional unit FU9 into the stack, and keep on the above process. FU10, FU11, and FU12 are pushed into the stack one by one. Now, the output of FU12 does not meet the objective F, and FU12 has no unvisited successor, so it is removed. As the new top, FU11 an unvisited successor FU13, so push FU13 into the stack. The output of FU13 meets the objective F, so output the FUs of the stack from the bottom to the top as another objective FUCS. And then, keep on testing the top of the stack. FU13, FU11, FU10, and FU9 all have no unvisited successor, so they are removed one by one. Now the stack is empty again, and there is no more other HFU, so the algorithm ends. There are three objective FUCSs generated in total.

3.2. Flow of the algorithm

Based on the assistant variables and basic principle, we built the computer program for the algorithm. Its flow diagram is shown as follows.

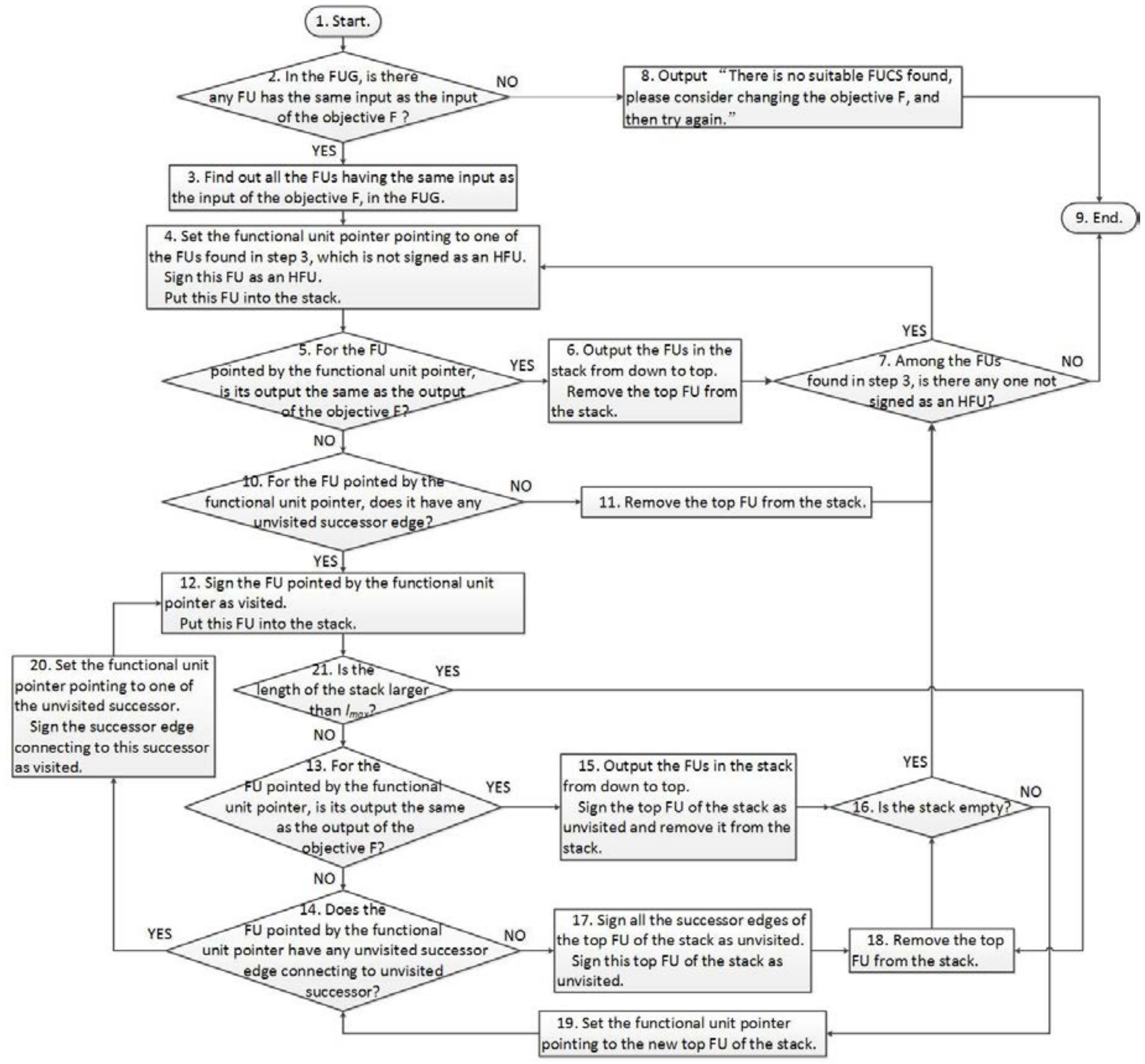

1. Start.
2. In the FUG, is there any FU has the same input as the input of the objective F ?
NO
8. Output "There is no suitable FUCS found, please consider changing the objective F, and then try again."
YES
3. Find out all the FUs having the same input as the input of the objective F, in the FUG.
9. End.
4. Set the functional unit pointer pointing to one of the FUs found in step 3, which is not signed as an HFU. Sign this FU as an HFU. Put this FU into the stack.
YES
5. For the FU pointed by the functional unit pointer, is its output the same as the output of the objective F?
YES
6. Output the FUs in the stack from down to top. Remove the top FU from the stack.
7. Among the FUs found in step 3, is there any one not signed as an HFU?
NO
NO
10. For the FU pointed by the functional unit pointer, does it have any unvisited successor edge?
NO
11. Remove the top FU from the stack.
YES
12. Sign the FU pointed by the functional unit pointer as visited. Put this FU into the stack.
20. Set the functional unit pointer pointing to one of the unvisited successor. Sign the successor edge connecting to this successor as visited.
21. Is the length of the stack larger than l_{max}?
YES
NO
13. For the FU pointed by the functional unit pointer, is its output the same as the output of the objective F?
YES
15. Output the FUs in the stack from down to top. Sign the top FU of the stack as unvisited and remove it from the stack.
YES
16. Is the stack empty?
NO
NO
14. Does the FU pointed by the functional unit pointer have any unvisited successor edge connecting to unvisited successor?
YES
NO
17. Sign all the successor edges of the top FU of the stack as unvisited. Sign this top FU of the stack as unvisited.
18. Remove the top FU from the stack.
19. Set the functional unit pointer pointing to the new top FU of the stack.

Fig.3 General flow diagram of the computer program of the algorithm

The detail steps of this program are shown as follows.

Step 1, start.

Step 2, in the FUG, determine whether there is any FU has the same input as the input of the objective F. If yes, turn to step 3, and if no, turn to step 8.

Step 3, find out all the FUs having the same input as the input of the objective F, in the FUG.

Step 4, set the functional unit pointer pointing to one of the FUs found in step 3, which is not signed as an HFU. Sign this FU as an HFU. Put this FU into the stack.

Step 5, for the FU pointed by the functional unit pointer, determine whether its output is the same as the output of the objective F. If yes, turn to step 6, and if no, turn to step 10.

Step 6, output the FUs in the stack from down to top. Remove the top FU from the stack.

Step 7, among the FUs found in step 3, determine whether there is any one not signed as an HFU. If yes, turn to step 4, and if no, turn to step 9.

Step 8, output "There is no suitable FUCS found, please consider changing the objective F, and then try again."

Step 9, end.

Step 10, for the FU pointed by the functional unit pointer, determine whether it has any unvisited successor edge. If yes, turn to step 12, and if no, turn to step 11.

Step 11, remove the top FU from the stack.

Step 12, sign the FU pointed by the functional unit pointer as visited. Put this FU into the stack.

Step 13, for the FU pointed by the functional unit pointer, determine whether its output is the same as the output of the objective F. If yes, turn to step 15, and if no, turn to step 14.

Step 14, determine whether the FU pointed by the functional unit pointer has any unvisited successor edge connecting to unvisited successor. If yes, turn to step 20, and if no, turn to step 17.

Step 15, output the FUs in the stack from down to top. Sign the top FU of the stack as unvisited and remove it from the stack.

Step 16, determine whether the stack is empty. If yes, turn to step 7, and if no, turn to step 19.

Step 17, sign all the successor edges of the top FU of the stack as unvisited. Sign this top FU of the stack as unvisited.

Step 18, remove the top FU from the stack.

Step 19, set the functional unit pointer pointing to the new top FU of the stack.

Step 20, set the functional unit pointer pointing to one of the unvisited successor. Sign the successor edge connecting to this successor as visited.

Step 21, determine whether the length of the stack is larger than l_{max}. If yes, turn to step 18, if no, turn to step 13.

4. Preliminary evaluation for the FUCSs

For the objective FUCSs obtained by the algorithm, they can be preliminarily evaluated and chosen.

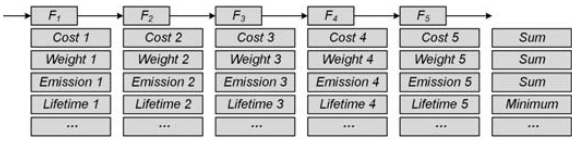


Fig.4 An example of the preliminary evaluation of an objective FUCS

As shown in Figure. 4, this FUCS is made of FU1-FU5. And each of the FUs has several features, like cost, weight, emission, life time and so on. Through these features, we can calculate the corresponding features of the whole FUCS. For example, the cost of the whole FUCS can be calculated by summing up the costs of all its FUs. And for its weight and emission, we can use the same method to calculate. And there is not only one method of summing features up. Like the life time of the FUCS, it can be calculated by finding the minimum of the life times of all its FUs.

In this approach, we don't give these features close restraints for the consideration of flexibility. This preliminary evaluation is not a kind of constraint, but just a preference for helping the designers to make the decision which alternative should be the chosen one.

5. Application and implementation

5.1. Design a kind of solar-powered wiper blades

Assume that we need to design a kind of solar-powered wiper blades. So, for the objective F, its input can be described by keyword "Light", and its output can be described by two keywords, i.e. "Reciprocating" and "swing". Now, assume the FUG is made up of 12 FUs, i.e. solar-battery, solar-heater, photographic-film, DC-motor, electric-heater, DC-light, electromagnet, crank-rocker, pulley-belt, gear-pair, cam, and crank-slider. So, the objective FUCSs should be generated with them.

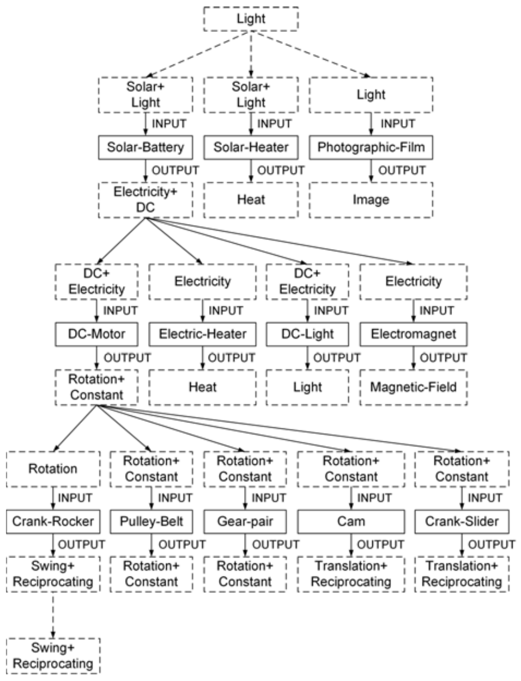


Fig.5 Illustration of the solar-powered wiper blades designing

As shown in Figure. 5, the input of the objective F is described by keyword "Light". So search the FUs whose input describing keyword is "Light" in the FUG. Three FUs can be found, i.e. solar-battery, solar-heater, and photographic-film. Their output describing keywords are {"electricity", "DC"}, {"heat"}, and {"image"}, all having no intersection with the output describing keywords of the objective F, {"Swing", "Reciprocating"}. So all these three FUs cannot meet the objective F alone, we need to keep on generating FUCS. The output describing keywords of both solar-heater and photographic-film have no intersection with the input describing keywords of the rest of FUs, which means both of these two FUs have no successor. So, they have to be given up.

As for the solar-battery, its output describing keywords are {"Electricity", "DC"}, having intersections with the input describing keywords of DC-motor, electric-heater, DC-light, and electromagnet. So these 4 FUs are all its successors. Testing their output describing keywords, they all cannot meet the objective F, so the FUCS needs to be extended. Among these 4 FUs, there is only DC-motor has successor, so we just use it in the next step.

Now, the end of the generating FUCS is DC-motor, its output describing keywords are {"Rotation", "Constant"}, having intersections with crank-rocker, pulley-belt, gear-pair, cam, and crank-slider. So these 5 FUs are all its successors. Their output describing keywords are {"Swing", "Reciprocating"}, {"Rotation", "Constant"}, {"Rotation", "Constant"}, {"Translation", "Reciprocating"}, and {"Translation", "Reciprocating"}, which means only the output describing keywords of crank-rocker meet the objective F. So there is an objective FUCS generated, solar-battery→ DC-motor→crank-rocker. As for pulley-belt, gear-pair, cam, and crank-slider, they all have no successor, so they should be given up, and there is no more other FU in the FUG, so the program ended with the one objective FUCS.

5.2. Design the functional module of a friction-abrasion testing machine

Additionally, if the design goal consists of several FRs or Fs, the design process should be divided into several corresponding parts. During every part of design process, just one FR or F can be considered, and an objective FUCS will be generated for it. Finally, all the objective Fs can be met by the corresponding FUCSs.

Assume that we need to design a friction-abrasion testing machine with four functional requirements, i.e. load module for producing the pressure between the two samples, driver module for producing the linear motion between the two samples, heating module for heating the samples, and measuring module for the measuring the target variables.

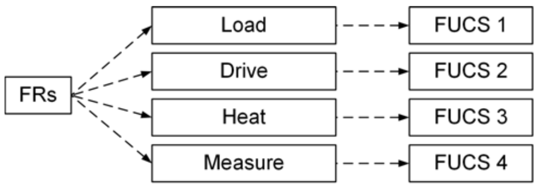


Fig.6 FRs of the target friction-abrasion testing machine

As shown in Figure. 6, there are 4 FUCSs need to be generated. Here, we just consider generation of the FUCS1 to save the paper length. Now, assume the FUG is made up of 10 FUs, i.e. DC-motor, electric-heater, DC-light, electromagnet, crank-rocker, pulley-belt, gear-pair, screw-nut, spring and crank-slider. The generating approach is the same as the case of the solar-powered wiper blades mentioned before, and the detail process is as shown in Figure. 7. The generated FUCS is DC-motor→screw-nut→spring.

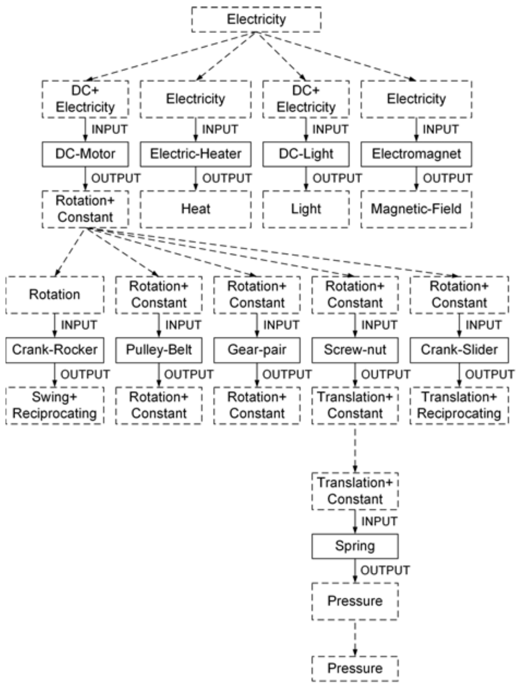


Fig.7 Illustration of the load module designing

6. Conclusion

Based on the hypothesis, definitions and model developed in PART 1, this paper established an algorithm for FUCS generation. FUs are treated as nodes, and connections of FUs are treated as edges, so the FUs in distributed resource environment and their connections construct the FUG. We applied DFS of graph theory to generate FUCSs from FUG. Based on this, we established the algorithm and built the corresponding computer program, and we also established an additional function to preliminarily evaluate the generated FUCSs. A kind of solar-powered wiper blades, and the load module of a friction-abrasion testing machine were designed as illustrations to prove the feasibility of this algorithm.

Acknowledgements

The authors are most grateful to the Key Project of State Key Laboratory of Mechanical System and Vibration (MSVZD201401), and the National Natural Science Foundation of China (51205247, 51575342).

References

[1] Mei J, Ren W, Ma G. Distributed containment control for Lagrangian networks with parametric uncertainties under a directed graph[J]. Automatica, 2012, 48(4): 653-659.
[2] Olfati-Saber R, Murray R M. Graph rigidity and distributed formation stabilization of multi-vehicle systems[C]//Decision and Control, 2002, Proceedings of the 41st IEEE Conference on. IEEE, 2002, 3: 2965-2971.
[3] Shen C C, Tsai W H. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion[J]. Computers, IEEE Transactions on, 1985, 100(3): 197-203.
[4] Gonzalez J E, Low Y, Gu H, et al. Powergraph: Distributed graph-parallel computation on natural graphs[C]//Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). 2012: 17-30.
[5] Cheung T Y. Graph traversal techniques and the maximum flow problem in distributed computation[J]. Software Engineering, IEEE Transactions on, 1983 (4): 504-512.
[6] Chen B, Xie Y B. A computer-assisted automatic conceptual design system for the distributed multi-disciplinary resource environment[J]. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 2016: 0954406216638886.
[7] Kwan M K. Graphic programming using odd or even points[J]. Chinese Math, 1962, 1(273-277): 110.
[8] Dijkstra E W. A note on two problems in connexion with graphs[J]. Numerische mathematik, 1959, 1(1): 269-271.
[9] López E B, Aucejo V C, Salvador Á C, et al. Problemas de rutas por arcos[J]. Questiió: Quaderns d'Estadística, Sistemes, Informatica i Investigació Operativa, 1983, 7(3): 479-490.
[10] Tarjan R. Depth-first search and linear graph algorithms[J]. SIAM journal on computing, 1972, 1(2): 146-160.
[11] Gregor D, Lumsdaine A. The parallel BGL: A generic library for distributed graph computations[J]. Parallel Object-Oriented Scientific Computing (POOSC), 2005, 2: 1-18.
[12] Barooah P, Hespanha J P. Graph effective resistance and distributed control: Spectral properties and applications[C]//Decision and control, 2006 45th IEEE conference on. IEEE, 2006: 3479-3485.