谢友柏设计科学研究基金项目
年度报告

项目名称：分布式资源环境中产品设计功能知识管理与集成方法研究

负 责 人：胡亚红

依托单位：浙江工业大学

通讯地址：杭州市西湖区留和路 288 号

邮政编码：310023

电子邮件：huyahong@zjut.edu.cn

电　　话：13073638970

报送日期：2023.04.25

1. 年度计划研究内容

2022.1-2022.12

(1) 收集与项目研究内容相关的文献，完成已有研究方法和研究结果的调研；

(2) 设计实现产品功能表示的预处理算法，以获取产品功能的标准化表示；

(3) 分析功能知识单元表示的特点，设计知识库的结构。

2. 年度研究进展及成果

2.1 年度研究进展

目前已收集了与项目研究内容相关的文献，并对已有的研究方法和研究结果进行了调研。在此基础上，本项目完成的主要工作包括：产品功能的标准化表示、基于 bucket 的用词对齐、功能知识图谱的设计、以及功能单元匹配算法。下面逐一介绍各部分研究工作。

（1）产品功能的标准化表示

功能单元是描述产品功能的最小单位，本项目采用“输入+输出”的方式对功能单元进行描述。因为 输出描述了产品所能对外提供的功能，因此规定一个功能单元可包含若干个输入，但仅仅包含一个输出。功能单元的输入和输出采用关键词表示，其中关键词又可分为核心词和修饰词，如图 1 所示。

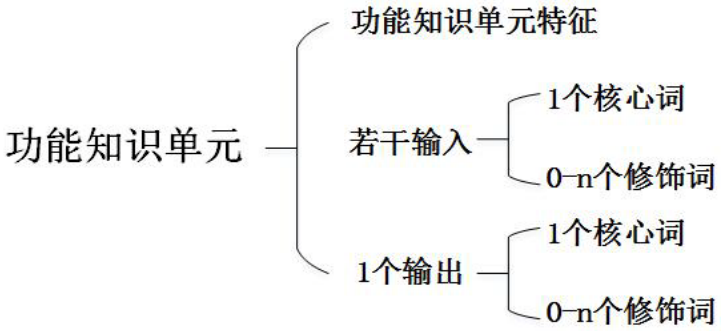

图 1 功能知识单元的表示

图 1 中，功能知识单元特征存放此单元重要的非功能要求，例如价格、交货期、可靠性等，采用关键词（包括名称及对应的值）表示，例如“价格（100元）”。输入和输出的表示方法相同，均采用核心词+修饰词的形式。修饰词是对核心词进行的可度量或不可度量的修饰。如某输入为“流动的水”，其中核心词为“水”，“流动的”是不可度量的修饰词。再如，对于输出“220V 的电”，“220V”则是对核心词“电”的可度量的修饰词。

（2）基于 bucket 的用词对齐

在使用关键词表示法对功能单元的输入/输出进行描述时，由于同义词的存在，不同人描述相同的输入/输出所使用的关键词有可能不同，这样会导致对功能单元间匹配关系的错误判断。例如，对于输入“electricity supply”和“electric power supply”，因为其核心词 electricity supply 和 electric power supply 是同义词，均表示电源，所以两者实际是对同一种输入的不同表述。然而，若单纯从单词是否相同去判断，并无法将两者等同。

在跨学科知识表示中，受不同学科背景的影响，这个问题尤为明显。为此，本项目提出了一种基于桶（bucket）的同义词管理机制来构建关键词词典，词义相同的词存储在一个桶中，每个桶用一个标签词（tag）表示。基于 bucket 的关键词词典生成步骤如下：

（a）当第一个单词被添加至词典时，为其创建一个桶，并为此桶设置标签词作为其唯一标识符，标签词的默认值为桶中添加的第一个单词，允许设计师对每个桶的标签词自由修改。

（b）当一个新单词被添加至词典时，根据这个单词与现有各桶的标签词之间的相似度，推荐与它最为相似的 N 个桶标签词。由设计师决定是将其分配给 Top-N 个推荐的桶中的一个，还是为其创建一个新桶。

建立关键词词典后，在描述输入/输出时，存储在同一个桶中的不同单词由同一个标签词代指，从而有效地避免表达的歧义。特别的，由于多义词同时存在于多个桶中，设计者需根据使用时的语境判断其桶的归属。依旧使用上述的例子进行说明，假设存在桶“electricity supply，electric power supply”，以 electricity supply 为标签词，那么不管设计师使用“electricity supply”或“electric power supply”描述功能的输入，此输入均会被自动转化为“electricity supply”后存储至知识库。

（3）功能知识图谱的设计

本项目使用知识图谱进行功能单元的存储，其中知识图谱中的节点表示功能单元，节点之间的边表示两个功能单元间的匹配关系。每个节点的信息如表 1 所示。

表 1 知识图谱中节点信息

属性	含义
ID	每个功能单元在功能知识图谱中唯一的标识
需求类型	功能单元所能够满足用户的需求的类型，可取值为物质需求、社会需求和精神需求
功能类型	功能单元所提供的功能的类别，可取值为转变功能、支承功能、存储功能和激励功能
输入	功能单元的输入，使用“核心词+修饰词”表示。输入可以有多

	个
输出	功能单元的输出，使用“核心词+修饰词”表示。输出仅有一个
实现载体	功能的实现载体，采用文字描述
特征	功能知识单元的特征，采用“关键词（包括名称及对应的值）”表示
输入匹配节点	是一个功能单元 ID 的集合，集合中的功能单元的输出与当前功能单元的输入匹配。
输出匹配节点	是一个功能单元 ID 的集合，集合中的功能单元的输入与当前功能单元的输出匹配

一个功能单元的输入/输出匹配节点决定了功能知识图谱中与此功能单元节点所邻接的节点。

（4）功能单元匹配算法

两个功能单元FU_i和FU_j匹配定义为：FU_i和FU_j的输入和输出间存在匹配关系。这里，一对输入和输出满足匹配关系指：它们的核心词存在于同一个 bucket 中，且输出的修饰词包含于输入的修饰词。

当一个功能单元进入知识图谱时，需要使用功能单元匹配算法确定它的输入匹配节点和输出匹配节点。功能单元匹配算法描述如下。

输入：功能单元FU_i的输入和输出信息

输出：FU_i的输入匹配节点集合和及输出匹配节点集合

（a） 将当前知识图谱中的所有功能单元放入待匹配功能单元集合 M 中。

（b） 当$M \neq \emptyset$时，从 M 中取出一个功能单元FU_j (i ≠ j)，执行：

i. 如果FU_j的输出与FU_i的输入匹配，则将FU_j的 ID 放入FU_i的输入匹配节点集合。

ii. 否则，判断FU_j的输入是否与FU_i的输出匹配。若匹配，则将FU_j的 ID 放入FU_i的输出匹配节点集合。

iii. 将FU_j从 M 中删除。

（c） 算法结束。

2.2 年度研究成果

本年度项目的研究成果包括申请专利一项，完成研究论文一篇（投往《Advanced Engineering Informatics》）。详细信息见附录。

附录：

国 家 知 识 产 权 局

310009

杭州市庆春路 9 号长堤明苑 22 层 B 座 杭州天正专利事务所有限公司

楼明阳(0571-87242818)

发文日：

2023 年 03 月 06 日

申请号：202310202577.X　　发文序号：2023030600582030

专 利 申 请 受 理 通 知 书

根据专利法第 28 条及其实施细则第 38 条、第 39 条的规定，申请人提出的专利申请已由国家知识产权局受理。现将确定的申请号、申请日等信息通知如下：

申请号：202310202577X

申请日：2023 年 03 月 06 日

申请人：浙江工业大学

发明人：胡亚红,蓝翔

发明创造名称：基于深度强化学习的面向创新设计的功能知识集成方法

经核实，国家知识产权局确认收到文件如下：

权利要求书 1 份 3 页,权利要求项数 ： 4 项

说明书 1 份 9 页

说明书附图 1 份 3 页

说明书摘要 1 份 1 页

发明专利请求书 1 份 4 页

实质审查请求书 文件份数：1 份

申请方案卷号：329610

提示：

1.申请人收到专利申请受理通知书之后，认为其记载的内容与申请人所提交的相应内容不一致时，可以向国家知识产权局请求更正。

2.申请人收到专利申请受理通知书之后，再向国家知识产权局办理各种手续时，均应当准确、清晰地写明申请号。

审 查 员：自动受理

联系电话：010-62356655

审查部门：初审及流程管理部

国家知识产权局 专利申查业务章 1101081359734

200101 2022.10

纸件申请，回函请寄：100088 北京市海淀区蓟门桥西土城路 6 号 国家知识产权局专利局受理处收

电子申请，应当通过专利业务办理系统以电子文件形式提交相关文件。除另有规定外，以纸件等其他形式提交的文件视为未提交。

Innovation design orientedfunctional knowledge integration framework based onreinforcement learning

Xiang Lan[a],Yahong Hu[a,*], YoubaiXie[b,c], XianghuiMeng[b], YilunZhang[b], Qiangang Pan[a]

[a] *College of Computer Science and Technology, Zhejiang University of Technology, 288 Liuhe Road, Hangzhou 310023, China*

[b] *School of Mechanical Engineering, Shanghai Jiaotong University, 800 Dongchuan Road, Shanghai 200240, China*

[c] *School of Mechanical Engineering, Xi'an Jiaotong University, 28 West Xianning Road, Xi'an 710049, China*

Abstract

According to the basic law of Design Science, new product design is based on existing design knowledge. Knowledge integration can be applied to product function design to shorten design timeand improve the design quality through effective use of the existing knowledge. With the increase of the product design complexity and the number of design knowledge, itis harder and harder for traditional traversal-based algorithms tocomplete knowledge integration under acceptable time cost. A Reinforcement Learning (RL) based functional knowledge integration framework is proposed. Thefunctional knowledge is represented by its input and output, and organized using a knowledge graph.The Q-network is constructed and trained for the deep Monte Carlo method-based functional unit chain generation algorithm. The performance experiments show that comparing with the traditional searching algorithms, the RL based algorithm can provide same quality design scheme with much shorter time. The proposed algorithm is promising to realize real-time functional knowledge integrationin large-scale knowledge bases.

Keywords: Functional knowledge integration; Computational design synthesis; Reinforcement Learning; Knowledge graph

1 Introduction

Innovative design is the key for companies to maintain market competitiveness. Conceptual design, as the initial stage of design, has always attracted much attention. According to the fundamental law of Design Science, new product design is based on existing design knowledge. Knowledge integration, or design synthesis, means to search and reuse existing design knowledge to assist new product design. Different from designers' traditional internal search methods such as brainstorming, knowledge integration can be combined with computer technology conveniently, and various computer-aided conceptual design scenarios see its application. During product design, it is widely accepted that the satisfaction of product function requirements is the primary consideration of designers [1,2,3,4]. Function design is the starting point for the following

[*]Corresponding author at: College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China
Email: huyahong@zjut.edu.cn

structure design and behavior design. Therefore, this paper focuses on the function level design synthesis, i.e., functional knowledge integration (FKI). In more detail, FKI is the search and combination of existing functional knowledge through automated algorithms to generate innovative solutions that meet the product’ s target function requirements.

In general, a function is described by its inputs and outputs. Knowledge for function design is stored in the knowledge base (KB) in the form of functional units (FU), the smallest reuse unit of

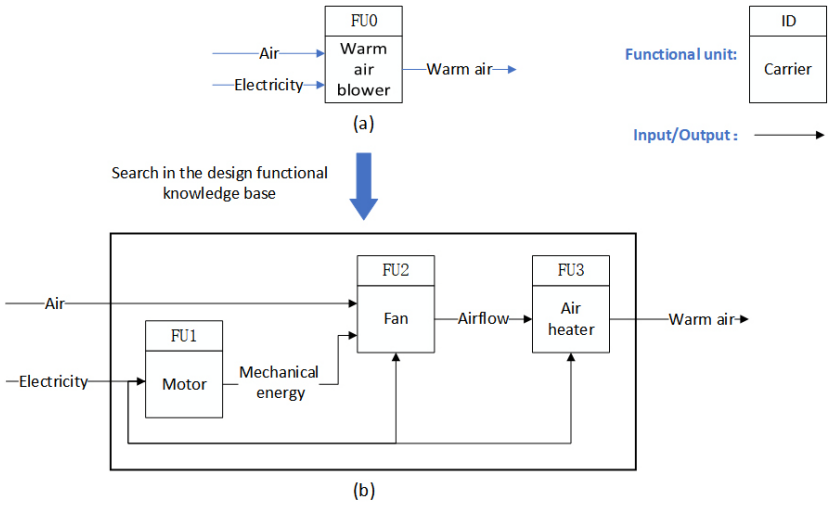

Fig. 1. An example of FKI.

function design knowledge.A simple example of the warm air blower design is shown in **Fig. 1**. The requirement, as well as the function of this product, is to provide warm air, and the inputs and output of the blower are given in **Fig. 1(a)**. FKI is introduced to find a possible way to design this product by searching a given knowledge base, and it finds that three functional units, i.e., FU1 (motor), FU2 (fan), and FU3 (air heater), are suitable. According to the inputs and outputs of these FUs, the design can realize the function of the warm air blower by linking them together, as shown in **Fig. 1(b)**.

Nowadays, the complexity of the design task is increasing day by day. An intuitive manifestation is that the knowledge required by lots of product design has changed from single-disciplinary to multi-disciplinary, and the useable design knowledge stores with different vendors. This trend has led to more general knowledge representation models and a significant increase in the size of design knowledge bases. The efficiency of FKI algorithms, as the guarantee for their effective execution in large-scale knowledge bases, has naturally become an important research direction.

Because the FKI problem shares many similarities with the graph search problem in essence, graph theory plays a pivotal role in the computerization of the FKI process.Helm and Shea [5] introduced object-oriented graph grammars to the traditional Function-Behavior-Structure (FBS) conceptual design model toimprove the representability and computer comprehensibility of FKI processes. For a particular FKI task of planar N-bar mechanisms design with rotary, prismatic, and pin-in-slot joints, Huang and Campbell [6]explicitly represented the planar topology of the mechanism as a graph to enumerate all possible topologies of the mechanism with any combination of the three joints. Münzer [7] proposed a graph-based and object-oriented representation of functional units. According to this representation, they provided a general and

automated FKI approach. Herber et al. [8] and Short et al. [9] further improved the enumeration capability of graph-based FKI approach through a tree searching algorithm, and achieved complete coverage of the solution space. Most of the above work adopted the top-down FKI method. Here, Top-down means if a function is too complex to be matched directly by a functional unit, the original function is disintegrated into sub-functions with decreased complexity or granularity. The sub-functions can be further divided into sub-sub-functions when necessary, andall these functions form a tree with the original function as the root. After the sub-functions represented by the leaf nodes are successfully matched, backtracking is carried out along the decomposition path to synthesize the design scheme of the root function. Commonly, such work builds on FBS design model [1] and flow-based input/output representation [2].

Chen and Xie[10,11] proposed a more flexible keyword-based representation of input/output, and reformulated the FKI process as a multi-source path searching problem. They also proposed an automated algorithm to generate functional design schemes by generating chain-shaped functional unit collections. The searching algorithm was extended to generate a more complexbranch-chain-shaped structure through the introduction of the auxiliary functional unit chain and incomplete matching [12]. In these works, the FKI problem is solved with the bottom-up method, which eliminates the explicit function decomposition process and searches the suitable functional units purely by their inputs and outputs. For a complex function, each search iteration can only complete part of the design, and after multiple iterations, the design scheme of the full requirements can be finally obtained. Compared with the top-down method, bottom-up method pays more attention to the complexity of the function unit structure itself rather than the granularity, and it has higher probability to obtain more flexible and creative design schemes. The algorithm proposed in this paper belongs to bottom-up category.

However, the basic idea of most existing functional unit searching algorithmsis traversal based, whether they are top-down or bottom-up. These algorithms can perform quite well with small number of functional units, and they can find the optimal solution due to their enumeration nature. With the increasing scale of knowledge bases, these algorithms are prone to have combinatorial explosion problems, i.e., they cannot find a solution within afeasible time. Tosolvethisproblem, Zhang et al. [13] proposed a presentation method of granular information in knowledge graphs, which integrated graph reasoning technology while retaining the knowledge layering mechanism and improved the algorithm search efficiencyfrom the perspective of optimizing the knowledge storage structure. To make full use of distributed computing resources and improve the algorithm efficiency, Chen [14,15] distributed the computing load to multiple processors for parallel searching. As these methods do not change the ergodic nature of the searching algorithms, the combinatorial explosion problem has not been eradicated.

Reinforcement learning(RL) achieves competitive performance in many large-scale and imperfect-information games, such as Star craft [16], DOTA [17], Mahjong [18], and DouDizhu [19,20], yet it has rarely been tried in FKI [21,22]. One contribution of this work is to bring RL to FKI tasksto bring some possible, active, and new development directions.

In this paper, a functional knowledge integration framework based on RL is proposed.The core of this framework is the functional unit chain generation algorithm called the Deep Monte Carlo Search (DMCS). The input of DMCS is the semantic description of the new product's target function requirements, and the output is the functional unit chain describing the design scheme (defined in **Section 3.4**).Based on the characteristics of RL, the FKI process in DMCS is

essentially a sequential reasoning process.Compared with the traditional searching process, the optimal solution can be obtained without traversing the solution space, thus avoiding the combinatorial explosion problem. At the same time, the reasoning speed only depends on the length of the reasoning path and is insensitive to the size of the knowledge base, then DMCS can maintain real-time search even on large-scale knowledge bases.

The structure of the paper is as follows. In Section 2, the models used in FKI are formally described. Section 3 explains the functional unit chain generation algorithm. In Section 4, the experiment setup and results are given in detail. Section 5 concludes the paper and lists future work.

2 Model Description

FKI is essentially a design search problem that takes place in the functional knowledge base. Therefore, the construction of KB is provided before the formal description of the FKI problem.

2.1 Representation of Functional Knowledge Graph

Since the relationship among functional knowledge units in KB can be modeled as a planar graph structure, it is very nature to introduce the concept of knowledge graph to the FKI tasks. We name this knowledge base the functional knowledge graph (FKG). This graph database structure is suitable for the subsequent knowledge reasoning process [13].

FKG is represented by a directed graph$G(V,E)$, where$V$ is the set of nodesin$G$and E is the set of edges.Node$v_i \in V$ represents a functional unit, and edge $e_{ij} \in E$ denotes the matching relationship between $v_i$and v_j.Knowledge in FKG is expressed as triples in the form of (functional unit, matching relation, functional unit). For example, $FU_i \rightarrow FU_j$ indicates that the output of the functional unit $FU_i$is matched by the input ofFU_j. FU_i is called the predecessor of FU_j,and FU_j is called the successor of FU_i.To improve the convergence of the searching algorithm (see in **Section 3.2.2**) and measure the optimality of the resulting path (see in **Section 2.4**), the number of unmatched inputs and outputs between adjacent nodes is further added to the matching relationship as an important attribute value, represented as a label on the edge.For example,$FU_1 \overset{0}{\rightarrow} FU_2$means that $FU_1$and FU_2match completely.

2.2 Representation of Functional Unit

The structure of the functional unit ontology is shown in **Table 1**, which contains seven attributes. *ID* is the unique identifier of the functional unit in the FKG. *Input* and *Output* represent the input and output of a functional unit, and they describethe basic contents of a product function. *Prior* and *Next* reflect the position(or adjacency) relationship of functional units in the FKG. The entries in *Prior* and *Next* are the predecessors and successors of the current functional unit respectively.*Category* refers to the category of the function provided by the functional unit,

including the transformation function, support function, storage function, and incentive function [23]. *Carrier* means the physical or technical prototype to implement the function of the functional unit in the real world. Among them, *Input* and *Output* are of the most importance, and the *Category* and *Carrier* will not be covered further since they are reserved for later structure design work.

Table 1
The construction of the functional unit ontology.

Attribute	ID	Input	Output	Prior	Next	Category	Carrier
Type	Int	Set	Set	Set	Set	String	Set

Table2
Representation of input and output.

Input/Output		**Type**	**Format**
Keywords	Core word	Int	Bucket
	Modifier	Set of int	{Bucket,...}
Features		Set of tuple	{(Name,Range,Unit),...}

2.2.1 Representation of Input/Output

The keyword-based representation is chosen to describe the input/output of functional units. Compared with the traditional flow-based representation, this method is more concise and flexible. Without the limitation to material flow, energy flow and information flow, the input/output of functional units can be distinguished accurately. Keyword-based representation is promising to representinterdisciplinary function design knowledge.

AFU can have a certain number of inputs and outputs, each of which is described by keywords and features, as shown in **Table 2**. The keywords adopt the form of “modifier + core word”, and there is only one core word and several modifiers for each keyword. The core word is a word or phrase representing an object, and the modifier is a word or phrase providingadditionalsemantic information to the core word. For example, for the input “fresh, clean + flowing water”, “flowing water” is the core word, and “fresh, clean”are the modifiers. The features adopt the form of “name + range + unit” to provide specific constraint information, e.g., “voltage + [220] + V” can be a feature of the input “electricity”, where “voltage” is the feature name, “[220]” gives the valuerange of this physical variable, and “V” is the corresponding unit of the feature of voltage.

2.2.2 Synonym Management of Keywords

Eachwordmayhave synonyms, soitis common todescribethesame input/output ofanFUusing

different keywords, which leads to misjudgment when doing the automatic matching. For example, the keywords “electricity supply” and “electric power supply” are synonyms and are different expressions of the same input. However, these two keywords cannot be matched if exact same words are required.

To solve the problem of diverse expression, a bucket-based synonym management mechanism is proposed to construct the keyword dictionary in which a bucket is a cluster that stores synonyms together. The bucket management algorithm works in the following steps.

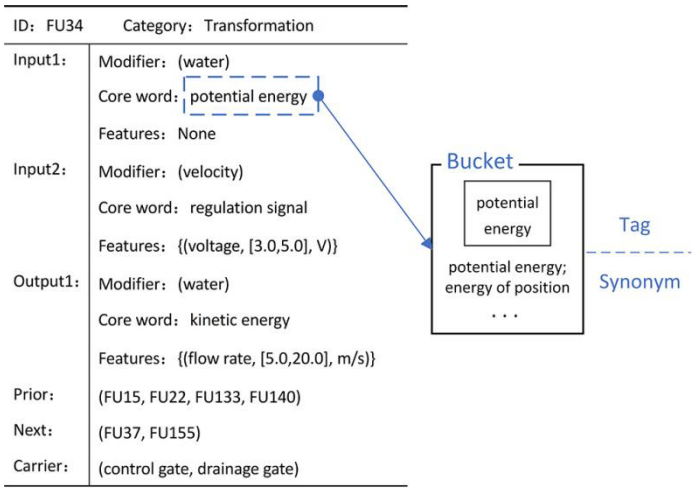

Figure 2. Example of function unit.

(1) When the first word is inserted into the synonym dictionary, a bucket is created. The bucket is assigned a tag as its unique identifier, and the first word into the bucket is the default value of this tag.

(2) When a new word is added to the dictionary, it is up to the user to decide whether to assign it to one of the existing top-n recommended buckets or create a new bucket for it, depending on its meaning. The recommendation is given based on the similarity between this new word and the tags of each bucket.

After establishing the keyword dictionary, when describing input/output, words in the same bucket correspond to the same tag. The use of buckets can avoid the occurrence of diverse expressions effectively. In particular, which bucket should a polysemy belongs to is determined by the designer according to the context. The example above is used for illustration again. If there is a bucket of “electricity supply, electric power supply” with “electricity supply” as its tag,then no matter whether the designer inputs the core word “electricity supply” or “electric power supply”, “electricity supply” will be stored actually in the KB after transformation.

Fig. 2 gives an example of a functional unit built according to the above rules.

2.3 Rules for Matching

There are two kinds of matching relationship considered in this paper, one is the matching between an input and an output, and the other is the functional units matching. The former means that an input and an output are regarded as equivalent to some extent. The latter means that these two functional units can be integrated to form a new bigger functional unit. Input/Output matching is the foundation for matching between corresponding functional units. The specific rules for matching are as follows.

Rules for Input/Output Matching. For a pair of input and output, they match if their core words are in the same bucket, the modifiers and features of the output contain those of the input. In particular, if a feature appears in both the input and output, the range of its value in input should cover that in output. **Fig. 3** shows an example of input-output matching.

Rules for Functional UnitsMatching. For functional units FU_i and FU_j, if at least one output of

FU_i matches at least one input of FU_j, $FU_i$is matchedbyFU_j.

2.4 Task of FKI

FKI is the task of searching for feasible solutions thatcan achieve the target function by exploring the knowledge base. For the convenience of algorithm description, the target function is abstracted

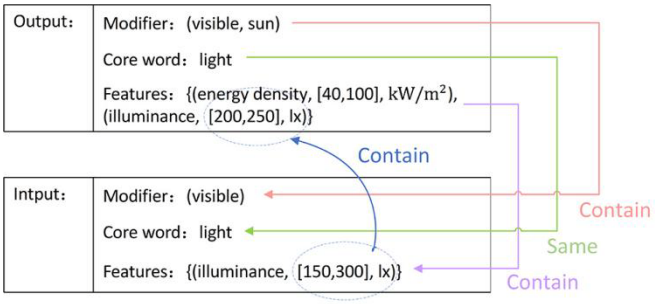

Fig.3. A matching pair of input and output.

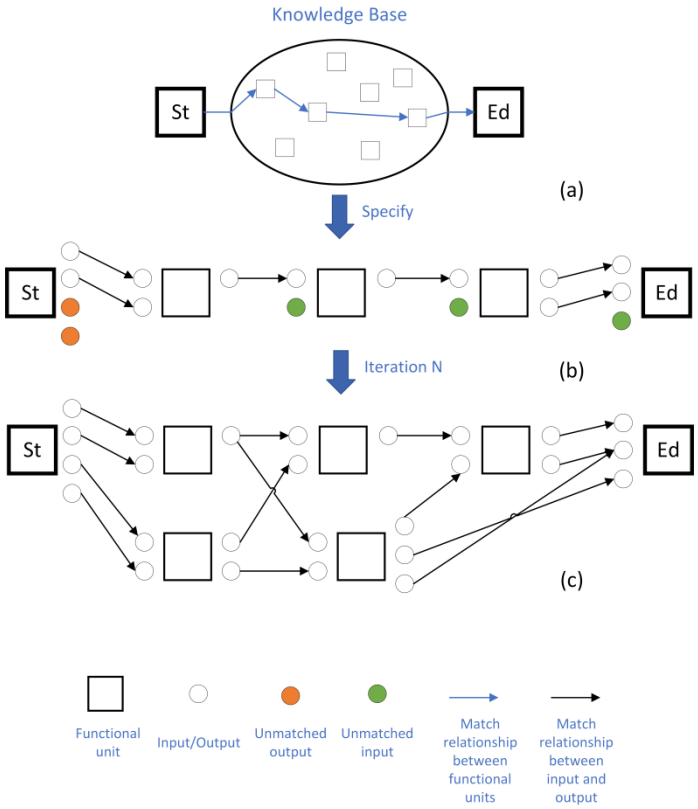

Fig. 4. The process of FKI task.

into two virtual nodes $St$and Ed that do not really exist in FKG. Node St only has the *Output* corresponding to the target inputs, and conversely node Ed only has the *Input* attribute corresponding to the target output.Then, an FKItask can be reformulated as searching for an optimal path in FKG from node St to Ed under certain constraints as illustrated in **Fig. 4(a)**.

Due to the complexity of the function structure, the exceptedresult of FKI is often a composite path with both series and parallel structures, as shown in **Fig. 4(c)**, which is difficult to be

obtained by a single search. A common solution is to decompose the FKI task into smaller subtasks by using the ideology of dynamic programming [24]. In particular, the success condition of each search is relaxed to find an optimal path from node St_n to Ed_n, where unmatched inputs and outputs are allowed, as shown in **Fig. 4(b)**. The nodes St_nandEd_ndenotes the current target outputandinput in iteration n. The inputs and outputs unmatched in search iteration n-1, which aremarked with green and red respectively in **Fig. 4(b)**, are taken as the target outputs/inputs for search iteration$n$. The number of unmatched inputs and outputs is recalculated after merging the resulting paths of iteration$n$-1 and iteration n. Theiteration continues until the number of unmatched inputs and outputs in the resulting path is 0, as shown in **Fig. 4(c)**, and then, the whole FKI task is completed.

Therefore, the FKI problem is an iterative problem. Since improving the efficiency of a FKI algorithm is the main objective of this paper, and the efficiency of iterative problem largely depends on the total number of iterations and the speed of each iteration, a greedy strategy is introduced to decrease the number of unmatched inputs and outputs in the path. In other words, the total number of unmatched inputs and outputs is the optimizationobjective for the result. The smaller this number, the better. This optimization object is in line with FKI's original optimality criterion of producing solutions as simple as possible. The efficiency of each iteration is another factor affecting the efficiency of the FKI algorithm. The searching algorithm for each iteration is called the Functional Unit Chain Generation algorithm, since it generates a series chain [24].

In general, the objective of an FKI is to find the optimal paths from node St to Ed in FKG. The basic idea is to decompose an FKI into subtasks and find the optimal chain for each subtask using the Functional Unit Chain Generation algorithm.As thetotal iterative framework (refer to **Section 4.1** for more details) issimple,this paper focuses on the functional unit chain generation algorithm.For a better description, the number of unmatched inputs and outputs between adjacent functional units is called the local redundancy number, and that in one path is the global redundancy number, respectively.

3 Functional Unit Chain Generation algorithm

3.1 Why RL

The process of RL belongs to a Markov Decision Process (MDP), as shown in **Fig. 5**. In any time step t, the agent performs the current optimal action A_t under the observed environment state S_t according to the policy π. Then the action transforms the environment state into S_{t+1}. Depending on the specific content of the transformation, the environment gives the agent different feedback, namely reward R_{t+1}. The agent adjusts its policy based on the reward, and a simple way is to perform the same action thatearned the largest rewardsbefore when facing the same state. The purpose of this adjustmentor the so-called learning is to maximize the cumulative rewards (return U_t) that may obtain in thefollowing time steps. Through such continuousinteractions with the environment, the agent gradually reinforces its understanding of the environmental rules, such as the mapping relationship of $(S_t, A_t, S_{t+1}) \rightarrow U_{t+1}$.

The reason to adopt RL to solve the FKI problem is for effectiveness and efficiency.

EffectivenessFor a general heuristic searching algorithm such as A*, whether it has good performance in a large graph database depends heavily on the heuristic function that measures the position relationship between the current node and the target node to guide the search direction. However, because of the asymmetric mapping between the inputs and outputs of the functional

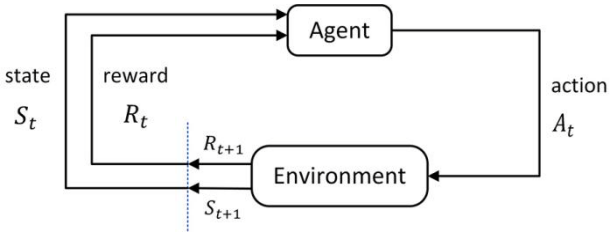

Fig. 5. The process of RL.

units, the position relationship between the non-adjacent nodes can hardly be measured in the FKI problem. Although the RL method is heuristic-based, it is essentially a trial-and-error learning method. Its policy optimization depends on the experience obtained from the agent and environment interaction, not a specific heuristic function.The lack of a complete position relationship between nodes causes the agent to get less feedback from the environment, which only delays the algorithm convergence and brings more training iterations. However, it has little effect on the algorithm's effectiveness.

Efficiency The RL method is more efficient than traditional traversal searching algorithms without considering the training time. It can obtain the approximate optimal path by sequential decision with no need to traverse the solution space, and it can decrease the effect of the FKG size on the algorithm efficiency.

3.2 Deep Monte Carlo Searching

The method proposed in this paper is an RL search algorithm based on Deep Monte Carlo (DMC), called Deep Monte Carlo Searching (DMCS).Referring to the description of the FKI task in **Section 2.4**, the FKG $G(V,E)$ is taken as the environment, and V constitutes theaction space$\mathcal{A}$.The episode of FKI is defined as follows.

Episode In the initial stage, nodes St and Ed (i.e.,thenodesintroducedtorepresent target functional requirements) are first specified or randomly generated (during network training). Starting from St, the agent selects and performs the action $a_t \in \mathcal{A}$at each time step t, corresponding to moving from the current node to the next node. If the agent reaches the node Ed within the specified maximum search length, thesearch is successful, and the action trajectory of the agent is the desired path; otherwise, the search is a failure. In this way, the search process changesto a sequential decision process.

The action selected at each time step in the episode depends on the policyπ. Usually, in off-policy RL, the greedy strategy $argmax_a Q(s,a)$ is adopted as the target policy π_t (the policy used for algorithm prediction and evaluation). Underπ_t, the action producing the largest Q-value among all state-action pairs (s,a) is selected as the decision result. It is clear that the optimal decision can be made as long as the Q-value of all actions in each state is known. Q-value is actually the return U mentioned before. Assuming that the value of$U$can be calculated by an

unknown function called Q-function, then how to obtain this Q-function is the primary consideration of RL methods. Actually, this is the selection of training methods.

In this paper, the training method is the Every-visit DMC [20], and its overall stepscan be summarized in the following cycle.

(1) Perform a complete episode based on the behavioral policy π_b. This policy is used only during algorithm training.

(2) Calculate the return U of each state-action pair (s,a) in the episode as the Q-value of the network.

(3) Update the network.

In detail, DMCuses Q-network as the approximation of the Q-function and trains it through Monte Carlo (MC) method. Q-network is a neural network to predicate Q-value. It receives the current observable environment state as input and outputs the Q-values of all candidate actions.MC method is a stochastic simulation method based on probability and statistics theory, and it is also a frequently-used solution in RL. Shortcomings of the MC method are known as being only effective for complete episodes and its low convergence due to high variance. FKI is an episodic-based task, i.e., each search process is independent, then MC method can work well under this situation. At the same time, as MC can be parallelized conveniently, it can generate multiple samples per second, and it is very efficient in wall-clock time. According to the research results of Zha et al. [20], the benefits brought by DMC in scalability are far higher than the adverse effects of high variance on algorithm convergence. In addition, FKI is a sparse reward task, i.e., the agent needs to go through a long list of states without feedback, and the only time step that produces a non-zero reward is at the end of the search. For the Temporal-Difference Learning (TD) algorithm, the Q-value in the current state needs to be estimated until the value in the next state is close to its true value [25,26]. Therefore, the convergence speed will be slowed down. However, DMC only calculates the real returns of each state after the end of the episode, so its convergence is not affected by the length of the episode. In conclusion, the DMC method is well suited for the FKI task.

3.2.1 Interval Epsilon Greedy Policy

Exploration-exploitation trade-off is the major consideration when generating behavior policy. Exploitation means that the agent always performs the optimal action based on the observed environment state, while the agent is allowed to try non-optimal action for more extensive interaction with the environment in exploration. To achieve greater long-term returns, short-term returns have to be sacrificed sometimes by giving up a certain number of exploitations in favor of exploration. The ε-greedy policy is defined in Eq.(1).

$$a_t = \begin{cases} \arg\max_a Q(s_t, a) & with\ prob\ (1-\varepsilon) \\ \text{random}\ action & with\ prob\ \varepsilon \end{cases} \tag{1}$$

where ε is a scalar between 0 and 1. In this policy, the agent exploits with probability $1-\varepsilon$ and performs random exploration with probability ε.

ε-greedy is one of the most basic and commonly used policies in RL.To improve the network performance, an interval ε-greedy is proposed, in which the value of εchanges dynamically based on the number of iterations, as shown in Eq.(2).

$$\varepsilon = \begin{cases} 1 - 0.1 \times \left\lfloor \dfrac{epi \times 10}{epi_{mid}} \right\rfloor, & 1 \le epi < epi_{mid} \\ 0.01 \qquad\qquad\qquad\quad , & epi_{mid} \le epi \le epi_{\max} \end{cases} \tag{2}$$

Whereepidenotes the current number of episodes，epi_{max} is the maximum number of thetraining episodes, andepi_{mid} is half the number of epi_{max} .The main idea of the interval ε-greedy policy is to divide the whole training process into two parts according to the number of training episodes. The agent is encouraged to explore with higher ε in the first half and exploit with lower ε in the second half. This simple method not only makes the global and local vision of the agent more balancedbut also makes the whole learning process smoother. Specifically, in this paper, the total iteration space is divided equally into two parent intervals. The former is further divided equally into ten subintervals, and the corresponding ε decreases uniformly from 1 to 0.1, The εin thelatterpart is fixed to 0.01.

3.2.2Rewards

The Rewards are the feedback from the environment for the transition of the observable states, and they can guide the agent's actions. In this paper, the rewards consist of four components: process, success, failure, and death reward.

(1) Success reward

The agent receives the success reward if it completes a successful search. It is an integer between 0 and 10 to measure the optimality of the resulting path. When the redundancy number of the resulting path is 0, the agent gets the maximum reward of 10. When the redundancy number is greater than or equal to 10, the path has no learning significance, and the agent gets the minimum reward of 0. The existence of this reward is to guide the agent to find successful paths with as few redundancies as possible. The successreward is calculated by Eq.(3).

$$r_{succ} = \max(0, 10 - redn_{\mathrm{g}}) \tag{3}$$

where $redn_g$denotes the global redundancy number.

(2) Process reward

In the task of FKI, because it is hard to measure the distance between any node and the target node Edin FKG, the transition of any two intermediate states cannot be evaluated by the change of their distance from Ed.

To further explain, assumee_a and e_b are any two connected entities in the knowledge graph, and the semantic information they carry in the form of word vectors is denoted as sem_a and sem_b, respectively. For the general semantic-based knowledge graph reasoning tasks, $\{e_a, e_b\}$ and $\{sem_a, sem_b\}$ are one-to-one mappings, as shown in **Fig.6(a)**. In this case, the distance between e_a and e_b can be approximated as the distance between sem_a and sem_b. For example, Liu et al. [27] proposed a dynamic reward mechanism, which correlated the reward with the cosine similarity between the current (entity, relation) pair and the target (entity, relation) pair. Their mechanism effectively alleviated the low convergence of the Actor-Critic network-based reasoning algorithm caused by the sparse reward. However, for the FKI task, a functional unit usually has multiple inputs and outputs, that is, $\{e_a, e_b\}$ and $\{\{sem_{a(i)}\}, \{sem_{b(i)}\}\}$ are one-to-one mapping, as shown in **Fig.6(b)**. Since the mappings between $\{sem_{a(i)}\}$ and

$\{sem_{b(i)}\}$are not clear, it's hard to measure the distance between e_a and e_b precisely by the word vector operation.

Since no suitable measuring method is available, the agent only gets non-zero rewards at the end of the search. In other words, the value of theprocess reward is always zero.

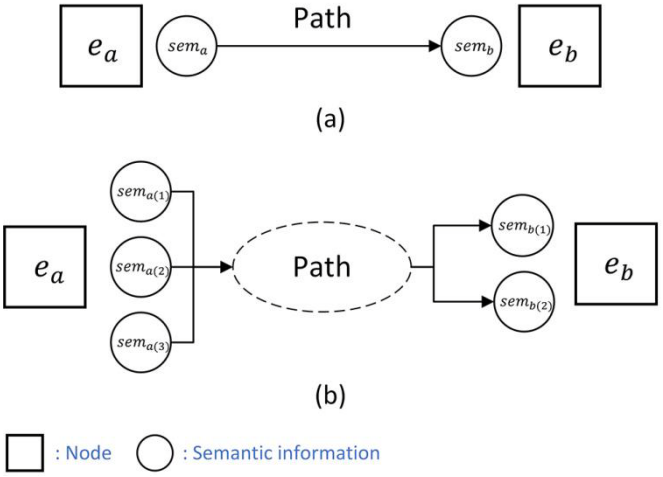

Fig. 6. The semantic mapping relationship between connected nodes in general inference task (a) and FKI (b).

(3) Failure reward

To go against the convergence reduction caused by the unobservable distance, a health-oriented hypothesis is proposed to explore the potential connections between the current path and the target node from another perspective.

Assume that the description of the functional requirements given by designers are accurate enough, which is normally true. It can be seen that the closer the functional unit chosen is to the target input, i.e., a smaller redundancy number of the current path, the more consistent the chosen FU is with the designer's idea. Then, we assert that the current path is more likely to lead to the target output, i.e., healthier.

Based on this hypothesis, the failure reward, a floating number from 0 to 1, is proposed to measure the health of the resulting path if the agent encounters a search failure. The failure reward can encourage the agent to explore a healthier path and is defined similarly to the success reward by Eq.(4).

$$r_{fail} = \max(0, 1 - \frac{redn_g}{10}) \tag{4}$$

It can be seen that when the path deviates seriously from the target functional requirement, the redundancy number will be relatively large, and according to the experiments, this number is greater than or equal to 10. It is important to note that health cannot reflect the distance between the current node and the target node.For example, there is a target node *Ed* that the agent reached at time step 20, and coincidentally, the redundancy number of the resulting path is 0 at both time step 1 and time step 19. The path got the same health degree at these two time steps, but the distance between the agent and *Ed* at time step 19 is significantly closer than that at time step 1. Therefore, the health is a kind of orientation information which is not strong enough to be used as a parameter of process reward.

(4) **Death reward** There is a special termination state not mentioned in the above episode

definition, i.e., the agent stops searching because there is no legal next node to go. This state is also a search failure, but as it is similar to the situation that the agent reaches a dead corner in the maze problem which should be avoided, we distinguish this reward from the failure reward and call it the death reward. The value of the death reward is 0.

Generally speaking, the rewards of the closer time step is more reliable thanthose of the far away time steps. Therefore, the return U is usually defined as the weighted sum of the rewards shown in Eq.(4).

$$U_{t=0:T} = \sum_{t=0}^{T} \gamma^t R_t \tag{4}$$

where $\gamma \in [0,1]$represents the discount factor, the smaller this factor, the more myopic of agents. In this paper, $\gamma$is set to 1, because the FKI task produces non-zero rewards only at the last time step, and the early node selection is also very important for the final reward generation.

3.2.3Input Features of Q-Network

The input features of the Q-networkare the concatenated representation of state and action, including the current node, the adjacency node, the local redundancy number between the current node and its successors, the action trajectory, the global redundancy number, and the target functional requirements.

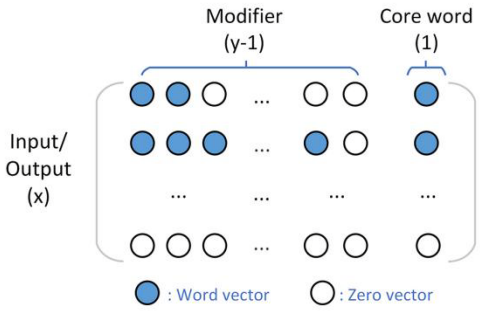

Figure 7. Example of encoding matrix of input and output.

Except that the two redundancy numbersare expressed in integer directly, other FU relevant states or actionsare represented by the*Input* and *Output*, which are encoded as an $x \times y$ matrix as shown in **Fig.7**.$x$is the maximum number of inputsoroutputs, and y denotesthe maximal number of keywords allowed. Each row in the matrix corresponds to one input/output, each element in column 1 to column $(y-1)$ corresponds to a modifier, and the last column corresponds to the core word. For each element in the matrix, if the corresponding keyword exists, its value is equaltothe word vector of the keyword; otherwise, it is the zero vector of the same dimension. To reduce the matrix size, an output is represented by the average value of the corresponding word vectors of all keywords when encoding action trajectory.

For theimplementation, the values of x and y are set as 12 and 6, respectively. ChineseEmbedding, an embedding dateset covers more than 12 million Chinese words and phrases released by Tencent AI Lab, is used to embed a keyword into a 200-dimensional vector. The complete input features are summarized in **Table 3**.

3.2.4 Architecture of Q-Network & Training algorithm

In order to ensure the efficiency of episode generation, the architecture of the Q-network is

designed as simple as possible, which consists of three components, i.e.,*Conv*, *LSTM* and *MLP*, as shown in **Fig.8**. The *Conv* part contains a convolution kernel with 200 channels and 12×6 size, and each convolution corresponds to a feature extraction of an input or output. The *LSTM* part is responsible for extracting sequential information in trajectory features, and it adopts theLSTM

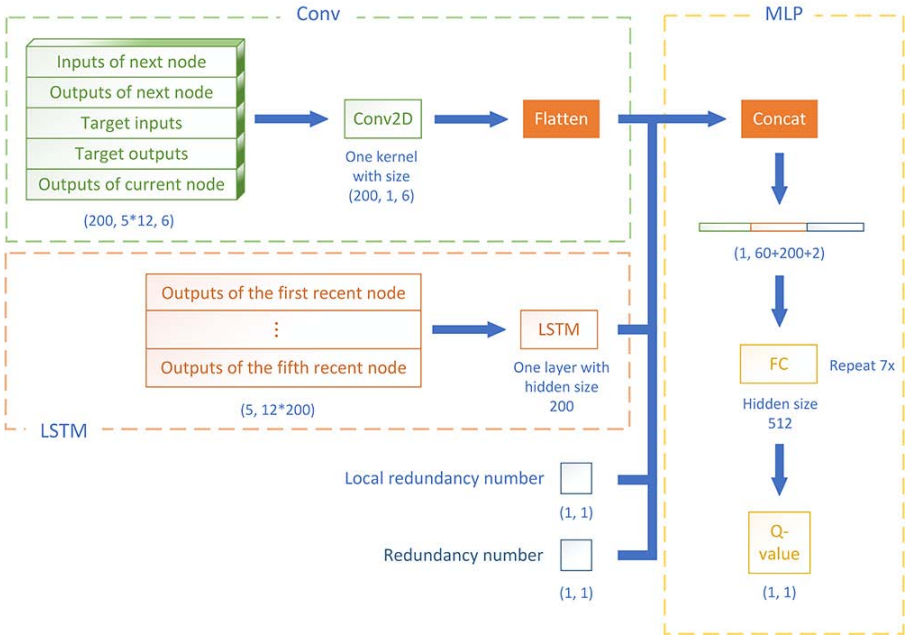

Fig. 8. Architecture of Q-Network.

network with a single layer, single direction, and a 200 hidden size. The *MLP* part contains 7 fully connected layers with a hidden size of 512, and its output is a vector of size $N \times 1$, where *N*represents the number of legal candidate actions or next nodes in the current state. For each inference of the network, the FU-relevant features are extracted by *Conv* and *LSTM*, then concatenated with the redundancy features to form a comprehensive feature. Based on these,*MLP* predicted the final Q-value.

A complete training algorithm description is shown in **Fig.9**. It follows the overall steps of Every-visit DMC and uses the Mean Square Error (MSE) as the loss function.

4Experiments

4.1 Experimental Setup

Most of the experiments in the literatureinclude only the effectiveness study of the algorithm [8,14,28]. They usually useone or a few specific real/well-designed design cases, including target functional requirements and known optimal paths, to demonstrate the algorithm execution details and validate the effectiveness. Algorithmperformance analysis using a large number of design casesis not considered. Currently, there are not enoughreal design cases available, and the scaleoftheKBs is not large. What's more, it is hard to integrate these independent small open-source Knowledge bases because of the different representations and storage structures of

FUs in each KB. Therefore, there are two dataset-level challenges for performance study.

(1) The scale of the existing knowledge bases is small, reflected both in the small number of functional units (size) and the small mean degree of the functional units (complexity). It is noteasy to compare the algorithm performance by the executing results of FKI tasks in such knowledge bases.

(2) There are few real design cases available in the existing knowledge base for testing the performance of the algorithms, so the experimental results have chanciness.

Table 3

Input features of Q-Network.

	Feature	**Size**
Action	Inputs matrix of the next node	200*12*6
	Outputs matrix of the next node	200*12*6
State	Matrix of target inputs	200*12*6
	Matrix of target outputs	200*12*6
	Outputs matrix of the current node	200*12*6
	Concatenated outputs matrix of the most recent 5 nodes	5*(12*200)
	Local redundancy number	1
	Redundancy number	1

Input: Learning rate ψ, exploration hyperparameter ϵ, maximum buffer size buf, experience playback threshold $rep_{threshold}$, maximum episode number epi_{max}
Initialize Q-networks Q, buffer B, temp buffer D, environment Env
for episode=1, 2,3, ... epi_{max} **do**
  for t=1, 2, 3, ... T do ▷ *Generate an episode*
    Observe s_t
    $\mathcal{A} \leftarrow$ Generate_Legal_Actions (s_t)
    $a_t \leftarrow \begin{cases} argmax_{a \in \mathcal{A}}\ Q(s_t, a), & with\ prob(1-\epsilon) \\ random\ \ action\ \ , & with\ prob\ \epsilon \end{cases}$
    Perform a_t, observe reward r_t
    Add $\{s_t, a_t, r_t\}$ to temp buffer D
  end for
  for t=T-1, T-2, ... 1 **do** ▷ *Obtain cumulative reward*
    $r_t \leftarrow r_t + r_{t+1}$ and update r_t in D
    Move $\{s_t, a_t, r_t\}$ from D to B
  end for
  if $B.size > rep_{threshold}$ **then** ▷ *Optimize network*
    Sample a batch of $\{s_t, a_t, r_t\}$ from B
    Update Q with MSE loss and learning rate ψ
  end if
  if $B.size > buf$ **then** ▷ *Maintaining buffer*
    Delete the earliest experience in B until $B.size \leq buf$
  end if
end for

Fig. 9. Training algorithm description of DMCS.

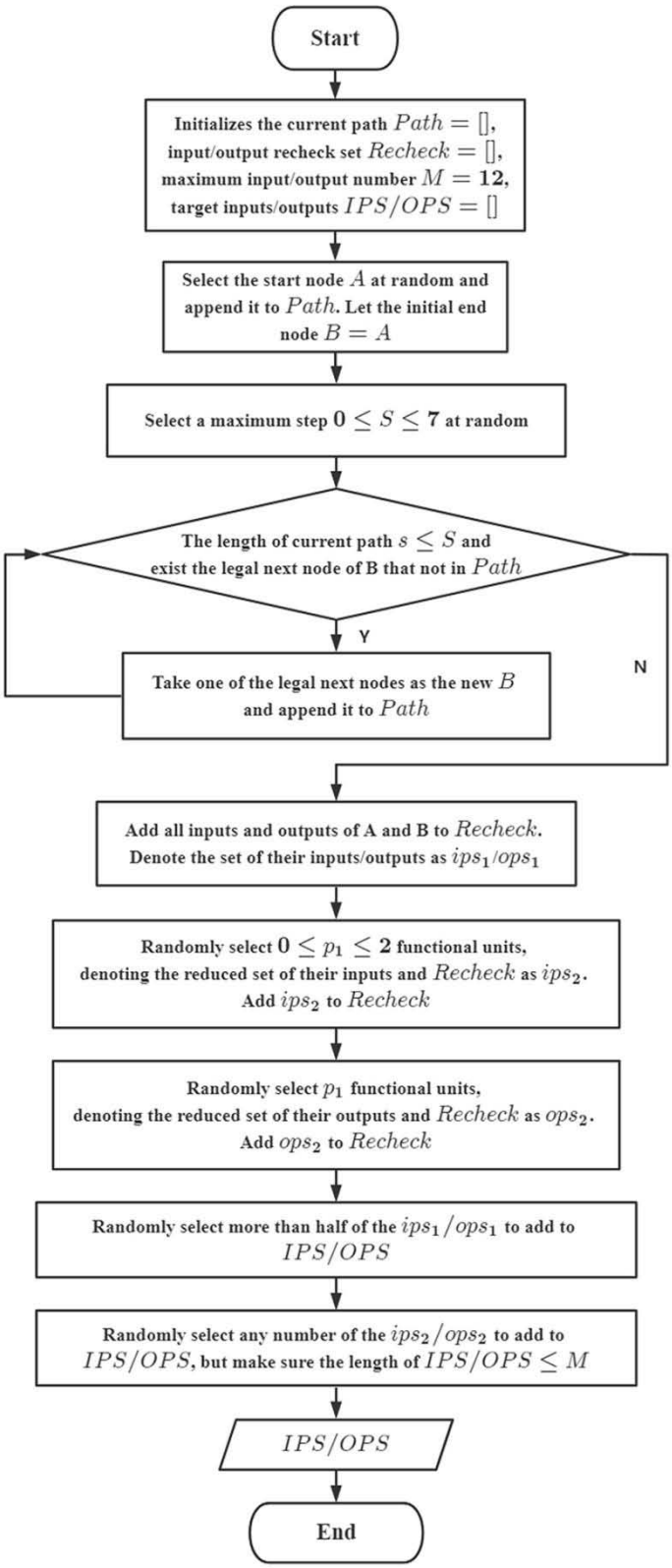

Fig. 10. The requirement generation algorithm.

4.1.1Requirement Generation Algorithm

There are two measures to evaluate the algorithm performance, one is the absolute performance, and the other is the relative performance. The absolute performance is obtained by comparing the resulting path with the optimal path, while the relative performance is measured by comparing the resulting paths obtained from different algorithms. The advantage of the relative performance is that the searching algorithms can be compared using generated target requirements instead of real-world cases.For challenge (2) just mentioned, the performance study in the paper is implemented by calculating the relative performance directly, rather than the traditional method of calculating the real performance first and then comparing the results.Thus, a requirement generation algorithm is provided to produce feasible target functional requirements randomly. Requirement generation is also the first step of FKI scenario generation. *Feasible*means that the requirements can be satisfied by the existing functional units in the knowledge base, i.e., there are paths from node SttoEd.

Since the target requirements should be satisfied by the existing functional units, the target inputs/outputs must be a recombination of the existinginputs/outputs. The core idea of the requirement generation algorithm is to randomly select functional units from the FKG and combine their inputs/outputs for the new target functional requirement. The main steps of the algorithm are summarized as below.

(1) Randomly select a node in the FKG as the start node A. Randomly generate a path starting from A with length z,a random integer, and use the last node on the path as the end node B. The inputs of $A$are put into set ips_1, and the outputs of B are inserted into setops_1.

(2) To enhance the complexity of thefunction structure, the inputs/outputs of the start/end node need to be expanded. Randomly choose$p$ functional unitsfrom the FKG, andput their inputs to set ips_2. Similarly, randomly choose$p$ functional units again from the FKG, andput their outputs to set ops_2. $p$is a random number.

(3) Let x be the maximal number of inputs/outputs of a functional unit and$y$ be a random number. Choose $y$elements in $ips_1$and no more than $x-y$elements in ips_2 to form the target input set. To maintain the basic characteristics of A, y should be larger than half of the size of ips_1. At the same time, to construct the target output set using the same method.

In this paper, $0 \le z \le 7$, $0 \le p \le 2$, $1 \le x \le 12$. It can be seen that according to the above algorithm, the starting point A and ending point Bmustbeconnected, and they can be used as $St$and Ed for a design requirement. **Fig. 10**shows the complete algorithm, including thepath loop detection, inputs and outputs de-duplication, etc.

4.1.2 FKGExpansion

Tohandle challenge (1), amanual-automatic hybrid method is used to expand the existing functional knowledge base, namely original FKG (FKG_0). In this part, the knowledge base provided by Chen [24] is taken as FKG_0. It contains59 FUs and 21 real cases, and the expansion shouldmeet the followingtwo requirements.

(1) Increase the size and complexity ofFKG_0.

(2) It should be guaranteed that the generation of functional units does not affect the original

real cases. It is not to separate the original functional units involved in the real cases from new generated functional units directly, as this will hinder the complexity of the design case from increasing with the expansion of the knowledge base.

Since a FU can be regarded as an implementation of the corresponding target functional requirement, i.e., a FUis corresponding to a functional requirement, the requirement generation algorithm provided in **Section 4.1.1**can generateFUs. In detail, a new FU can be generated by expanding existing inputs and outputs to any existing FU. In this case, the inputs and outputs of the new FU are from the existing FUs, so its degree is high, i.e., the above requirement (1) is satisfied.

One shortcoming of this method is that the new generated FUs are lack of logic and are not suitable as a part of a real case. To satisfy requirement (2), a new independent knowledge base is further built. The main steps of the whole expansion are as below.

(1) A new knowledge base $FKG_C$is built, which contains 59 new manually collected FUs. FKG_O and FKG_C are totally independent, i.e., the inputs/outputs of any two functional units from FKG_O and FKG_C are not matched.

(2) Generate three new functional units based on each functional unit in FKG_C, so the size of FKG_Cincreases by triple.

(3) Generate a new functional unit based on each functional unit in FKG_O as illustrated in **Fig. 11.**Denote the FU taken fromFKG_O as the basic FU, and the new FU after expansion as the mutation FU. During expansion, the inputs of the mutation FU is the same as those of the basic FU, and its outputs is expanded by outputs selected randomly from FKG_C. A unidirectional connection $FKG_O \rightarrow FKG_C$is established.

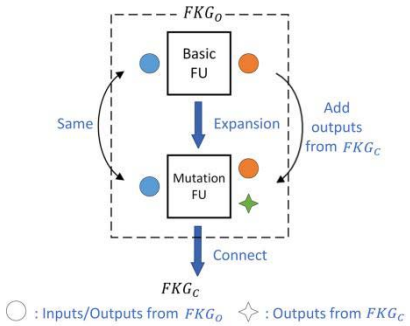

Fig.11. Special way of functional unit generation in FKG expansion.

A simple proof that mutation FUsdo not affect the original optimal paths of the real cases is as follows. Taking any node and its adjacent nodes in the optimal path as an example, since the inputs of the mutation FU is unchanged compared with the basic FU, its left adjacent node is unchanged. The output of the mutation FU increases, and it is connected to FKG_C. Because of the nonexistence of connection $FKG_C \rightarrow FKG_O$, the extra outputs can not lead to end node Ed, so the right adjacent node remains the same. Compared with the mutation FU, the basic FU is always selected as a better node because of lower local redundancy, so the current node is unchanged. In conclusion, any node and its adjacent nodes in the optimal path change nothing after expansion, and the relationship is established between the cases and these two knowledge bases.

(4) The final unified knowledge base, namelyFKG is obtained by merging FKG_OandFKG_C.

The scale comparison of knowledge base before and after expansion is shown in **Table 4**. The experiments in the following sections are conducted basedon FKG.

Table 4

The scale comparison of FKG before and after expansion.

	Function units (number)	Triples(number)	Complexity(mean degree)	Cases(number)
FKG_0	59	52	1.76	21
FKG	**354**	**6066**	**34.27**	21

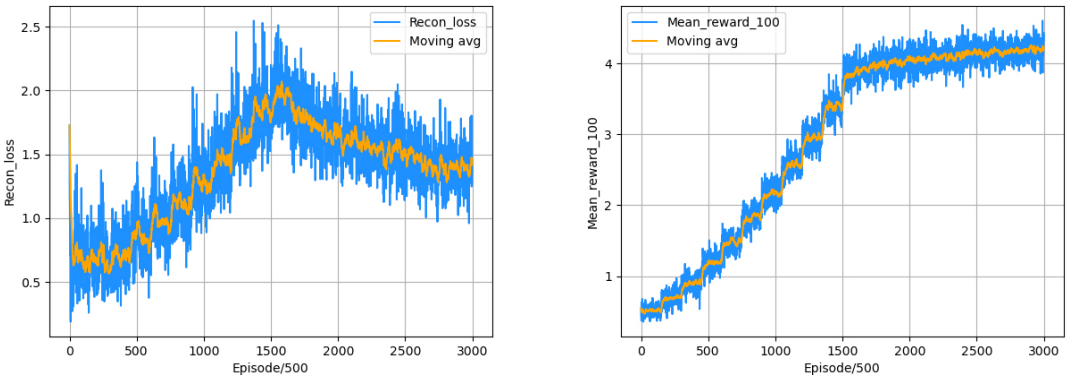

Fig.12. Loss (a) and mean (b) reward w.r.t. the number of training episodes where the current number of episodes is divided by 500 and the mean reward is calculated base on the most recently 100 episodes.

4.2 Implementation Details & Training Condition

DMCS algorithm is implemented based on Pytorch deep learning framework, ReLU activation function for MLP layers, Adam optimizer, and MSE loss function. The training completes in three days on a server with 20 processors of the 12th Gen Intel(R) Core(TM) i7-12700KF CPU @ 3.61GHz and a NVIDIA GeForce RTX 3080 10GB GPU, and the convergence curve is shown in **Fig.12**. The values of each hyperparameter are as follows: maximum search length $L = 10$, learning rate $\varphi = 5e - 7$, maximum episode number $epi_{max} = 1500000$, batch size $bat = 512$, experience replay threshold $rep_{threshold} = 1536$,and maximum buffer size $buf = 20000$.

The losses throughout the training process is shown in **Fig.12.(a)**. At the beginning of the training, the policy is learned from scratch, so there is a short period of rapid decline in losses. Then the agent enters the exploration period, and the policy is updated frequently, so the curve has a large fluctuation and shows a reverse growth. With the continuous decrease of ε, the policy tends to be stable, and then the losses decline to convergence.

The rewards during the training process is shown in **Fig.12.(b)**. Because of the interval epsilon greedy policy, the rewards increased regionally as the ε decreased regionally. The overall trend of the rewards in each region is to rise first and then flatten out, indicating that the training in each region is sufficient. Eventually, the rewards number converges at around 4.3.

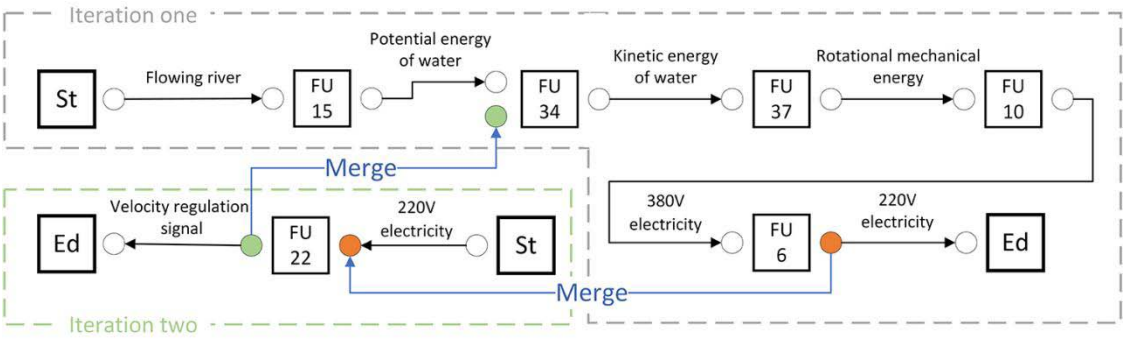

Fig.13. Demonstration of FKI process in Case one. A complete description of these FUs is available in the **Appendix**.

4.3 Effectiveness & Case Study

To verify the effectiveness of DMCS in solving the FKI problem, we executed DMCS on the 21 real design cases mentioned in **Section 4.1.2**, and the results are shown in **Table 5**. The results show that DMCS successfully finds the optimal path in all cases and prove that DMCS can handle the FKI problem.

To illustratehow DMCS works, the first case listed in **Table 5** is chosen to demonstrate the complete FKI process, including the iterative process of multiple functional unit chain generation.

Background Ina leisure center, there is a river flowing through.To achieve sustainable development, a device is needed to accomplish the hydropower generation to support the electrical appliances in the leisure center.

Target requirements According to the requirement of this product, the target input is "flowing river" and the target output is "220V + electricity".

Execution of DMCS

Iteration one Run DMCS with the target requirements. Path 1, the obtained optimal path, or to say, the functional unit chain, is shown in the dashed box labeled "Iteration one" in**Fig. 13.**In this path, the input "velocity regulation signal"of $FU34$ isnot matched. So, the self-matching is performed.Each unmatched input/output is checked to see if it can be matched by any other existing input/output in the path. After self-matching, this input still cannot be matched. Since it is the first iteration, no path merging operation is required, and the current path is the result of this iteration.

Iteration twoAfter the first iteration, the target inputs/outputs of the following iteration are defined as the unmatched outputs/inputs of the previous iteration by default. As the health-oriented hypothesis mentioned, accurate target requirements are conducive to achieve a suitable path for DMCS. At the same time, it is good to amend the existing design direction by the product designers. Therefore, designers are allowed to add or remove the inputs/outputs from the default target requirements. In this design case, the default target output is "velocity regulation signal", and there is no default target input, so "220V + electricity" is manually added to the target input according to the correlation.

Run DMCS with the revised target requirements, and the optimal path (Path 2) is achieved as shown in the box labeled "Iteration two" in **Fig. 13**. Since there is no unmatched input or output in Path 2, the self-matching is passed.

Now there are two optimal paths, i.e., Path 1 and Path 2, and it is necessary to merge them. The target input and output used in Iteration two connect with nodes $FU34$ and $FU6$ of Path 1 by Node$FU22$ in Path 2, and the relationship is shown with lines labeled Merge in **Fig. 13**. The merged path is taken as the result of this iteration. More generally speaking, path merging is performed between the path obtained from the previous iteration and the current path, denoted $path_{i-1}$ and $path_i$ respectively. The target requirement is first updated, i.e., for the target requirement of $path_i$, if there are inputs or outputs that do not exist in $path_{i-1}$, add them to the corresponding target requirement of $path_{i-1}$. And then, the matching relationship is updated, i.e., each node in $path_i$ excluding St and Ed is checked whether it matches any nodes in $path_{i-1}$or not. If so, this node is inserted to $path_{i-1}$.

As there is no unmatched input or output in the current iteration path, the algorithm ends and this path is taken as the final result.

During the process, all mentioned keywords are converted to the corresponding bucket tag by default. In this case study,not too many features of the inputs/outputs areconsidered. For some inputs/outputs with the same keywords but different features, part of their features are retained as modifiers to be added to the keywords for discrimination. For example, the modifiers “220V” and “380V” in “220V + electricity” and “380V + electricity” are essentially preserved voltage features.

To sum up, the overall framework of the proposed FKI method includes five parts, i.e., default requirement determination, requirement modification, functional unit chain generation, self-matching and merging. The flowchart is shown in**Fig. 14**.

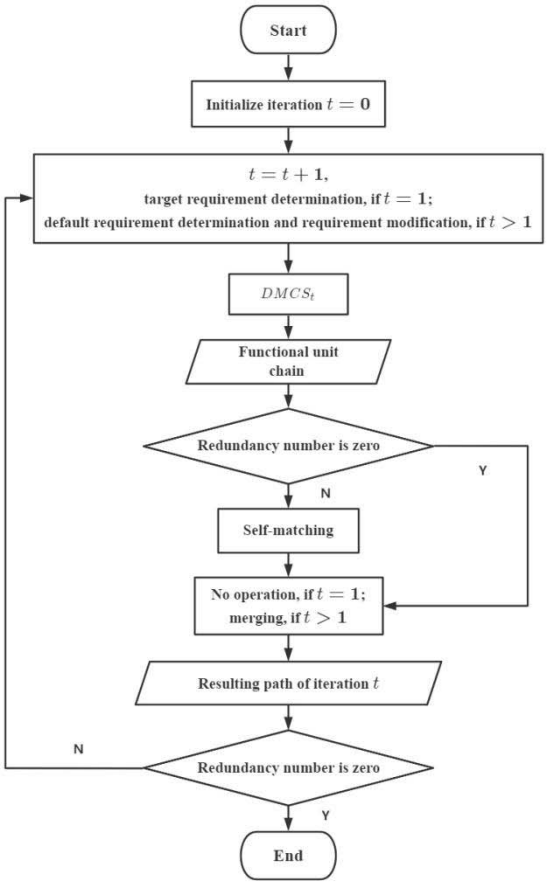

Fig.14. The overall framework of FKI.

Table5

Result of effectiveness study where the value of success and optimal metrics are bool.

Task	Target	Given	Success	Optimal
1	$St \xrightarrow{0} FU15 \xrightarrow{0} FU34 \xrightarrow{1} FU37 \xrightarrow{0} FU10 \xrightarrow{0} FU6 \xrightarrow{0} Ed$	$St \xrightarrow{0} FU15 \xrightarrow{0} FU34 \xrightarrow{1} FU37 \xrightarrow{0} FU10 \xrightarrow{0} FU6 \xrightarrow{0} Ed$	**1**	**1**
2	$St \xrightarrow{2} FU39 \xrightarrow{0} FU5 \xrightarrow{1} FU17 \xrightarrow{0} FU38 \xrightarrow{0} FU23 \xrightarrow{0} FU28 \xrightarrow{1} Ed$	$St \xrightarrow{2} FU39 \xrightarrow{0} FU5 \xrightarrow{1} FU17 \xrightarrow{0} FU38 \xrightarrow{0} FU23 \xrightarrow{0} FU28 \xrightarrow{1} Ed$	1	1
...	...	...	...	...
13	$St \xrightarrow{0} FU35 \xrightarrow{0} Ed$	$St \xrightarrow{0} FU35 \xrightarrow{0} Ed$	1	1
...	...	...	...	...
20	$St \xrightarrow{2} FU44 \xrightarrow{0} FU52 \xrightarrow{1} FU57 \xrightarrow{0} Ed$	$St \xrightarrow{2} FU44 \xrightarrow{0} FU52 \xrightarrow{1} FU57 \xrightarrow{0} Ed$	1	1
21	$St \xrightarrow{3} FU45 \xrightarrow{0} Ed$	$St \xrightarrow{3} FU45 \xrightarrow{0} Ed$	1	1
Mean	-	-	**1.0**	**1.0**

Table6

Performance comparison between algorithms.

	Success(%)	Lower redn(%)	Lower redn(within 3)(%)	Mean redn	Mean time (s)	Total time (s)
DMCS	**90.40**	**74.56**	**92.92**	6.13	**0.02**	**22.33**
DFS	17.50	-	-	3.14	104.65	265814.17
DMCS(without r_{fail})	86.30	36.96	50.41	15.76	0.03	28.72

4.4 Performance & Ablation Study

To evaluate the performance of DMCS, the functional unit chain generation algorithm proposed by Chen [24] is selected as the counterpart. Chen's algorithm is DFS (depth-first-search)based and referredto as DFS in this paper. In DFS, pruning is realized by memorization search, i.e., recording the paths from the visited node to the endnode Ed. However, this approach often leads to memory overflow during large-scale search tasks, even if only the optimal paths in different lengths are recorded. In order to ensure that the algorithm can run under the experimental conditions, the recorded content in each visited node is modified to be within a minimum step, or to say, if the distance between a node and the current node is beyond this minimum step, there will be no solution.

DMCS and DFS both run on 1000 target functional requirements generated randomly by the algorithm mentioned in **Section 4.1.1**. For each run, since it is well known that the maximum search length greatly affects the time efficiency of traversal algorithm, it's set to 8 as short as possible based on the possible lengths of the optimal paths, and the maximum search time is set to 300s, as ample as is acceptable. The experimental results are shown in **Table 6**. The performance

of these two algorithms is mainly compared from twoaspects.

Computational efficiency The computational efficiency of the algorithmsare reflected by three metrics, i.e.,*success*, *mean time* and *total time*. *Success* denotes the probability that the algorithm completes the search within the maximum search time and gives the optimal path successfully, that is, the probability of successful search. *Mean time* denotes the mean searching time of the successful searches. *Total time* denotes the total time taken by the algorithm to complete all the search tasks.

Results quality The quality comparison of the search results of the two algorithms is also reflected by three metrics, i.e.,*lower redn*, *lower redn (within 3)* and *mean redn*. *Lower redn* denotes the probability that the redundancy number of the optimal path obtained by DMCS is less than or equal to that of the corresponding DFS solution.Here, the DFS solution refers to the path with the minimum redundancy number found by DFS within the maximum search time, not necessarily the optimal path. This metric is counted only when the DMCS reaches a successful search. *Lower redn (within 3)* is defined on the basis of *lower redn*, and it denotes the probability that the redundancy number of the optimal path obtained by DMCS is less than or equal to the redundancy number plus 3 using DFS. *Mean redn* denotes the average redundancy number of the optimal paths obtained by DMCS or DFS solutions.

The experimental results are shown in **Table 6**. It is clear that DMCS only sacrifices a small part of the resulting quality (doubledthe value of *mean redn*) in exchange for a much higher computational efficiency than DFS (about 4 times the *success*increase and 5,000 times the *mean time* reduction). This result reveals the characteristics of these two algorithms. DMCS gets the approximate optimal solution directly by inference, while DFS gets the real optimal solution by traversing the search space. Therefore, it is foreseeable that if the scale of the knowledge base continues to expand, the quality gap of the results between DMCS and DFS will be smaller and smaller through adequate training, and the difference in computational efficiency will be bigger and bigger. Overall, DMCS is promising to achieve real-time FKI on ever-expanding design knowledge bases with acceptable optimality.

To validate the effect of the health-oriented hypothesis on improving algorithm convergence, an ablation experiment was designed. It is a version of DMCS without r_{fail} reward, and after training, this new model runs under the same experimental conditions as the previous experiments. And according to the results, the performance of the DMCS without r_{fail} reward is muchlower than that of DMCS except for the *mean time* and *total time.* This result is inlinewiththeexpectationbecause the redundancy number is used to measure path optimality besidesthe health-oriented hypothesis. Therefore, the absence of r_{fail} reward not only affects the *success*, but also haseffect on the results quality. In conclusion, r_{fail} reward designed on health-oriented hypothesis can lead to a much better convergence quality of DMCS.

5 Conclusion & Future work

For facilitating the new product design, the integration algorithm of existing functional knowledge units is illustrated through a series of work, including knowledge base construction, episode definition, reward setting, and Q-Network design.

We built a complete application framework of RL in FKI, and the DMCS algorithm is the kernel of this framework. DMCS has quite good computational efficiency in the real-time

generation of functional unit chains in large-scale KBs compared with the traditional traverse-based searching algorithms. DMCS supports an optional automatic or semi-automaticiterative FKI by allowing designers to modify the existing functional design scheme and subsequent design direction during the design process.

For future work, considering the need for further refinement of the conceptual design scheme, we will make full of the $Carrier$ attribute of the functional unit to record more information such as price, weight, volume, and the life cycle of a product. Based on this information, the designer can choose the appropriate specific design prototype for each abstract function in the scheme according to the requirements. The current evaluation of the optimal path considers only the redundant number, and a joint metric should be defined in the future. Last but not least, we will try more methods to continue improving the performance of the RL algorithm, such as parallelization, trying other neural architectures and balance strategies between exploration and exploitation.

Funding

This work is supported by the XieYoubai Design Science Foundation (No. XYB-DS-202205).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Gero, J. S. (1990). Design prototypes: A knowledge representation schema for design. *AI Mag.*, 11, 26-36. https://www.researchgate.net/publication/30870218_Design_Prototypes_A_Knowledge_Representation_Schema_for_Design

[2] Stone, R. B., & Wood, K. L. (2000). Development of a functional basis for design. J. *Mech. Des.*, 122 (4), 359-370. https://doi.org/10.1115/1.1289637

[3] Tang, L. (2008). An approach to function identification in automated conceptual design of mechanism systems. *Res. Eng. Des.*, 19 (2-3), 151-159. https://doi.org/10.1007/s00163-008-0048-z

[4] Kurtoglu, T., Swantner, A., & Campbell, M. I. (2010). Automating the conceptual design process: From black box to component selection. *Artif. Intell. Eng. Des. Anal. Manuf.*, 24 (1), 49-62. https://doi.org/10.1017/S0890060409990163

[5] Helms, B., & Shea, K. (2012). Computational synthesis of product architectures based on object-oriented graph grammars. *J. Mech. Des.*, 134 (2), 021008. https://doi.org/10.1115/1.4005592

[6] Huang, W., & Campbell, M. I. (2015). Automated synthesis of planar mechanisms with revolute, prismatic and pin-in-slot joints. In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 57137. American Society of Mechanical Engineers, V05BT08A062. https://doi.org/10.1115/DETC2015-46073

[7] Münzer, C. (2015). Constraint-Based Methods for Automated Computational Design Synthesis of Solution Spaces (Doctoral Dissertation). ETH-Zürich, Nr. 22990. https://doi.org/10.3929/ethz-a-010603411

[8] Herber, D. R., Guo, T., & Allison, J. T. (2017). Enumeration of architectures with perfect matchings. *J. Mech.*

Des., 139 (5), 051403. https://doi.org/10.1115/1.4036132

[9] Short, AR., DuPont, B.L., Campbell, M.I. (2019). A comparison of tree search methods for graph topology design problems. In: *International Conference on-Design Computing and Cognition*, pp. 75-94. Springer, Cham. https://doi.org/10.1007/978-3-030-05363-5_5

[10] Chen, B., &Xie, Y. B. (2017). A computer-assisted automatic conceptual design system for the distributed multi-disciplinary resource environment. *Proc. Inst. Mech. Eng. C*, 231 (6), 1094-1112. https://doi.org/10.1177/0954406216638886

[11] Chen, B., &Xie, Y. B. (2017). Functional knowledge integration of the design process. *Sci. China Technol. Sci.*, 60 (2), 209-218. https://doi.org/10.1007/s11431-016-0236-8

[12] Chen, B., &Xie, Y. B. (2018). A function unit integrating approach for the conceptual design synthesis in the distributed resource environment. *Proc. Inst. Mech. Eng. C*, 232 (5), 759-774. https://doi.org/10.1177/0954406217692008

[13] Zhang, Y., Wang, H., Zhai, X., Zhao, Y., & Guo, J. (2021). A C-RFBS model for the efficient construction and reuse of interpretable design knowledge records across knowledge networks. *Syst. Sci. Contl Eng.*, 9 (1), 497-513. https://doi.org/10.1080/21642583.2021.1937373

[14] Chen, B., Hu, J., Qi, J., & Chen, W. (2021). Concurrent multi-process graph-based design component synthesis: Framework and algorithm. *Eng. Appl. Artif. Intell.*, 97, 104051. https://doi.org/10.1016/j.engappai.2020.104051

[15] Chen, B., Hu, J., Chen, W., & Qi, J. (2021). Scalable multi-process inter-server collaborative design synthesis in the Internet distributed resource environment. *Adv. Eng. Inform.*, 47, 101251. https://doi.org/10.1016/j.aei.2021.101251

[16] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., … Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575 (7782), 350-354. https://doi.org/10.1038/s41586-019-1724-z

[17] Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H. P. d. O., Raiman, J., … Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. http://arxiv.org/abs/1912.06680

[18] Li, J., Koyamada, S., Ye, Q., Liu, G., Wang, C., Yang, R., Zhao, L., Qin, T., Liu, T.-Y., & Hon, H.-W. (2020). Suphx: Mastering Mahjong with deep reinforcement learning. http://arxiv.org/abs/2003.13590

[19] Jiang, Q., Li, K., Du, B., Chen, H., & Fang, H. (2019). DeltaDou: Expert-level Doudizhu AI through self-play. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 1265-1271. https://doi.org/10.24963/ijcai.2019/176

[20] Zha, D., Xie, J., Ma, W., Zhang, S., Lian, X., Hu, X., & Liu, J. (2021). DouZero: Mastering DouDiZhu with self-play deep reinforcement learning. http://arxiv.org/abs/2106.06135

[21] Campbell, M. I., Cagan, J., &Kotovsky, K. (1999). A-Design: An agent-based approach to conceptual design in a dynamic environment. *Res. Eng. Des.*, 11 (3), 172-192. https://doi.org/10.1007/s001630050013

[22] Vale, C. A. W., & Shea, K. (2003). A machine learning-based approach to accelerating computational Design. In: *International Conference on Engineering Design*, pp. 183-184. https://www.designsociety.org/publication/38/DS+31%3A+Proceedings+of+ICED+03%2C+the+14th+International+Conference+on+Engineering+Design%2C+Stockholm

[23] Xie, Y. B. (2018). Design Science and Design Competitiveness [M]. Beijing: Science Press. ISBN: 9787030554383. https://book.sciencereading.cn/shop/book/Booksimple/show.do?id=B662BB6AD54C311E0E053020B0A0ACF1C

000
[24] Chen B. (2019). Theoretical And Methodological Research On The Functional Knowledge Integration In The Design Process Based On The Distributed Resource Environment [D]. Shanghai:Shanghai Jiaotong University. https://doi.org/10.27307/d.cnki.gsjtu.2018.000634
[25] Szepesvári, C.(2010). Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 9. https://doi.org/10.1007/978-3-031-01551-9
[26] Beleznay, F.; Grőbler, T. &Szepesvári, Cs. (1999). Comparing Value-Function Estimation Algorithms in Undiscounted Problems (TR-99-02) , Technical report, Mindmaker Ltd. , Budapest 1121, Konkoly Th. M. u. 29-33, Hungary
[27] Liu, H., Zhou, S., Chen, C., Gao, T., Xu, J., & Shu, M. (2022). Dynamic knowledge graph reasoning based on deep reinforcement learning. *Knowl.-Based Syst.*, 241, 108235. https://doi.org/https://doi.org/10.1016/j.knosys.2022.108235
[28] Ma, J., Hu, J., Zheng, K., & Peng, Y. H. (2013). Knowledge-based functional conceptual design: Model, representation, and implementation. *Concurr. Eng. Res. Appli.*, 21 (2), 103-120. https://doi.org/10.1177/1063293X13487358

Appendix

Table 7lists all the FUs mentioned in the case study.

Table 7

A complete description of the FUs mentioned in **Fig. 13**.

ID：FU6	**Category：Transformation**
Input1:	Modifier：None
	Core word：electricity
	Features：{(voltage, 380, V), (phase number, 3, None), (AC/DC, AC, None), (power, [50, 150], kW)}
Output1:	Modifier：None
	Core word：electricity
	Features：{(voltage, 220, V), (frequency, 60, Hz), (phase number, 3, None), (AC/DC, AC, None), (power, [50, 150], kW)}
Prior:	(FU9, FU10, FU127, FU128)
Next:	(FU7, FU16, FU22, FU24, FU32, FU39, FU44, FU45, FU50, FU59, FU125, FU134, FU140, FU142, FU150, FU157, FU162, FU163, FU168, FU177)
Carrier:	(transformer)
ID：FU10	**Category：Transformation**
Input1:	Modifier：(Rotational)
	Core word：mechanical energy
	Features：{(rotational speed, [900,4000], r/min), (torque, [11,25], N.m)}

Output1:	Modifier：None
	Core word：electricity
	Features：{(voltage, 380, V), (phase number, 3, None), (AC/DC, AC, None), (power, [95, 105], kW)}
Prior:	(FU7, FU37, FU125, FU155)
Next:	(FU6, FU124)
Carrier:	(alternating-current generator)

ID：FU15 Category：Transformation

Input1:	Modifier：None
	Core word：flowing warter
	Features：{(flow velocity, [0.8,2], m/s), (flow rate, [0.65,1.87], m^3/s)}
Output1:	Modifier：(water)
	Core word：kinetic energy
	Features：{(maximum storage depth, 4, m)}
Prior:	None
Next:	(FU34, FU152)
Carrier:	(dam)

ID：FU22 Category：Transformation

Input1:	Modifier：None
	Core word：electricity
	Features：{(voltage, 220, V), (frequency, 60, Hz), (phase number, 3, None), (AC/DC, AC, None), (power, [50, 150], kW)}
Output1:	Modifier：(velocity)
	Core word：regulation signal
	Features：{(voltage, [3,5], V)}
Prior:	(FU6, FU12, FU124, FU130)
Next:	(FU34, FU152)
Carrier:	(velocity regulator)

ID：FU34 Category：Transformation

Input1:	Modifier：(water)

	Core word：potential energy
	Features：None
Input2:	Modifier：(velocity)
	Core word：regulation signal
	Features：{(voltage, [3,5], V)}
Output1:	Modifier：(water)
	Core word：kinetic energy
	Features：{(flow velocity, [5,20], m/s)}
Prior:	(FU15, FU22, FU133, FU140)
Next:	(FU37, FU155)
Carrier:	(control gate, drainage gate)

ID：FU37	**Category：Transformation**
Input1:	Modifier：(water)
	Core word：kinetic energy
	Features：{(flow velocity, [4,25], m/s)}
Output1:	Modifier：(Rotational)
	Core word：mechanical energy
	Features：{(rotational speed, [1000,3000], r/min), (torque, [11,20], N.m)}
Prior:	(FU34, FU152)
Next:	(FU10, FU128)
Carrier:	(hydraulic turbine)