Journal of Information Hiding and Multimedia Signal Processing ©2018 ISSN 2073-4212
Ubiquitous International Volume **9**, Number **4**, July **2018**

Optimizing Software Development Requirements based on Dependency Relations

Zhixiang Tong, Xiaohong Su, Xiao Ding, Jiaxin Lin
School of Computer Science and Technology,
Harbin Institute of Technology
Harbin 150001, China

Received September, 2017; revised May, 2018

Abstract. *Software development planning and the optimization of requirements are increasingly complicated due to the complex technical and functional dependency relations among software development requirements. In this article, a directed graph model is adopted to describe the dependency relations among requirements from a global perspective. The concepts of value and cost were introduced into this model to describe resource limitations and employer expectations during the development of a software system. The resultant requirement sets are evaluated according to the ratio of the software value to its costs (i.e., value-to-cost ratio). Furthermore, we design a heuristic function which could consider one requirements' value form the global view, and propose an A*-pruning algorithm based on the A* algorithm to obtain the requirement sequence set of a global optimum. The results generated with this approach are compared with those of five other methods under various goals and constraints. Experimental findings indicate that the A*-pruning algorithm can outperform other methods in terms of value-to-cost ratio. This work therefore provides an optimization scheme for requirement analysis in software engineering.*
Keywords: Requirement dependency, Directed graph, A*-pruning algorithm, Requirement optimization

1. **Introduction.** Software development is essentially a process of constructing a software system based on user requirement analysis findings. A clear understanding of user requirements for software functions and performance is a prerequisite to the successful implementation of a software project[1]. The CHAOS Report of the Standish Group listed the success rates of projects from 1994 to 2012, and problems in requirement engineering were determined to be the primary reason for project failure[2]. According to a study conducted by the European Software Institute, 50% of companies regarded requirement specification and requirement management as the most significant challenges in system development[3]. The rapid and accurate acquisition of core requirements to save development costs and satisfy employer requirements is a fundamental issue in software development. As the scale of a software project expands continuously, the problem of repetitive and overlapping requirements increases in severity. For example, the developers of the FBI Virtual Case File project, which cost 1.7 million dollars, held joint application design meetings for almost six months before finally establishing a requirement specification of approximately 800 pages[4]. The security consultant of the project indicated that the project requirement document was too bloated to highlight the essential requirements; under such circumstances, identifying the requirements that must be satisfied and

Table 1. Classification and description of common dependency relations

Category	Dependency	Relations
strong constraint	"And" dependency	**Requirement R1 can be achieved if and only if Requirement R2 is achieved**
strong constraint	"Or" dependency	**Requirements R1 and R2 are contradictory and cannot be achieved simultaneously.**
strong constraint	Precedence dependency	**If Requirement R1 is to be achieved, then Requirement R2 should be selected first.**
Weak constraint	Value-related	**If Requirement R2 is achieved, then its values for employers depend on whether or not Requirement R1 is met.**
Weak constraint	**Cost-related**	**If Requirement R2 is achieved, the implementation cost is related to whether or not Requirement R1 is achieved.**

prioritizing them with proper selection methods are key steps in the development of a complex software project.

In the process of identifying software requirements, many dependencies are detected among the requirements. According to the Tracking of the Projects Requirements established by IBM developer Works, the major reason for the failure of a project is the unsuccessful management of requirements[5]. When mistakes are detected and solved late, the costs of correcting these errors increase. Therefore, a developer should guarantee the traceability of software requirements, including the source of these requirements, reasonability, realization conditions, and the influence of requirement changes, at the development planning and requirement optimization stage. The processes of analyzing and managing the dependency relations among these requirements are referred to as requirement interaction management (RIM)[9, 10]. RIM can directly affect the selection of requirements, traceability management, evolutionary process, and the importance attached to these requirements[11].

Previous work can rank the priorities of requirements, e.g., requirement A is of higher priority than requirement B[13]. However, this optimization model may be improper or erroneous if requirement B precedes requirement A. For example, employers generally have two basic requirements for an E-commerce platform, namely, online searches for commodities and online commodity transactions. From the perspective of a merchant, the latter should be prioritized; however, without the feature of online search for commodities, online transactions can hardly be conducted. Aside from the precedent relation, many other dependency relations are detected, including the and/or relation, complex correlations, and dependency in consideration of costs, values, technology, structure, and functions. Table I lists several dependency relations.

With the existence of dependency relations among requirements, the priority selection problem is certainly complex[68], which involves several formidable challenges: (1) formalizing the dependency relations among various requirements; (2) taking the overall considerations of the priority order and dependency relations of the requirements as well as balancing the conflicts between these factors; (3) incorporating real-life constraint conditions into requirement priority selection (e.g., constraints on development resources, such as resources and costs, as well as the expectation of employer benefits); (4) determining an effective dependency management method for these requirements to meet diverse requirements and thus enhance the management efficiency.

To solve the above issues, we propose the A*-pruning algorithm, a path finding algorithm based on a heuristic function. The primary contributions of this study are summarized as follows. (1) A method is proposed for converting the requirement dependency relation graph to an activity-on-vertex (AOV) network. The requirement optimization problem, which is nondeterministic polynomial-time hard (NP-hard), can thus be converted into an optimal path search problem on the graph. (2) On the basis of the concept of heuristic value in path finding algorithms, the A*-pruning algorithm is established so that the optimal solution set can be determined in polynomial time. (3) The experimental results obtained for a large requirement data set indicate that the proposed method outperforms five other counterpart techniques. The remainder of the article is organized as follows. Section II presents the background of the study and related works. Section III describes the modeling and solving processes of dependency relations with the A*-pruning algorithm. Section IV discusses the quantitative evaluation of our approach and presents the analysis results. Finally, we close with conclusions in Section V.

2. **Related Work.**

Requirement optimization is a popular research area, and much research has been conducted in this regard. Early solutions developed include the analytic hierarchy process [32], quality function deployment [33], cumulative voting (100-Dollar Test) [34], and requirement- prioritization approaches [12]. However, the requirement priority must be manually annotated and then democratically ranked using these methods; that is, these methods are relatively subjective. Moreover, these methods do not consider the different interests of users in determining requirements or the dependency relations among requirements [13–15].

Donald et al. [16] described the necessity of dependency relations among requirements and pointed out the major challenges in the process of requirement priority selection. These challenges include numerous requirement items, the restrictions of limited resources and requirement quality, requirement conflicts among different users, requirement changes, and incompatible requirements. Nevertheless, these researchers did not propose an automatic processing method, and continued to adopt the human-mediated modeling method of guiding requirement optimization. Greer et al. [17] used the genetic algorithm (GA) to iteratively solve the optimization issue; however, this method is difficult to converge and is highly unpredictable. Moreover, the results are highly dependent on the quality of the objective function. Therefore, the GA-based method is improved by a modified multi-objective evolutionary algorithm [18]. However, the dependency relations among requirements remain neglected. An increasing number of works focus on the dependency relations among requirements. These studies can be classified into two categories as follows.

The first category defines the dependency relation. For example, Zhang et al. [19] determined five fundamental relations among the requirements that involve and, or, precedence, value-related, and cost-related relations. The first three relations are used as parameters in applicability equations, whereas the latter two are applied as performance parameters. Li et al. [20] defined four fundamental relations, that is, task dependency, goal dependency, resource dependency, and soft-goal dependency. Luo et al. [21] highlighted six fundamental relations, namely, calling, interrupting, informing, arousing, revising, and resource control. These studies have presented the dependency relations of requirements from different perspectives.

The second category emphasizes the solving of requirement priority problems on the premise of dependency relation. Carlshamre et al. [22] combined linear programming techniques with the interdependency relations among requirements to model requirement prioritization and selection. Ruhe and Saliu proposed the combination of computational intelligence and artificial judgment into integer linear programming to solve the conflicts between these factors [23]. Van den Akker et al. improved on the linear programming-based method proposed by Ruhe and Saliu and maximized the benefits under limited budgets [24, 25]. Bagnall et al. considered the precedence dependency among requirements and used a directed acyclic graph to describe this dependency relation [26]. This work was then improved by Greer and Ruhe by including the and dependency in the model. Consequently, the precedence and and dependencies are both adopted as constraint conditions to guide the optimization search [27]. To represent the interdependency relations among requirements, Moisiadis et al. [28] developed a requirement prioritization tool based on either the binary matrix or the tree structure. Laurens [29] proposed an identification method and a modeling approach to determine unnecessary requirement dependencies. Alebrahim et al. [30] presented a problem-based requirement dependency analysis method and constructed a problem graph with requirements based on domain knowledge. The requirements can be simplified by utilizing weak dependencies.

The aforementioned methods are limited in the following ways. Concretely, linear programming is merely a single-objective algorithm and can handle only constraint conditions that can be transformed into either equations or inequalities. Search-based requirement interaction approaches can address only the precedence and and dependencies among the requirements. These approaches merely define relations and provide weight-allocation methods; the interaction approaches do not solve the problem of requirement prioritization with multiple dependency relations from a global perspective. Some methods classify requirements through clustering and allocate weights for different categories of requirements. However, the samples are insufficient and may not resemble the actual situation because practical conditions and industrial peculiarities are neglected.

To overcome the limitations of previous works, we propose an A*-pruning algorithm for requirement optimization. Unlike prior methods, the A*-pruning algorithm introduces additional constraint conditions into requirement priority selection (e.g., constraints on development resources, such as resources and costs, as well as the expectation of employer benefits). This algorithm also designs a heuristic function that can assess potential value-to-cost ratio. Through the overflowing-pruning process, the value-to-cost ratio of the entire system can be maximized under the constraints of dependency relations among requirements so that employer resources can be reasonably utilized.

3. A*-Pruning Algorithm.

3.1. Overview.

The problem of requirement optimization is selecting an appropriate subset from the universal set to optimize value-to-cost ratio. Given a requirement set consisting of n requirements, the number of corresponding subsets is 2^n. Under the condition of requirement dependency, the requirement may fall into a local optimum instead of a global optimum. Furthermore, the optimality of a solution set cannot be verified in polynomial time, i.e., this problem is NP-hard. In fact, the requirement optimization problem under the condition of requirement dependency can be converted into a node expansion problem in an AOV network. To describe the problem formally, we construct a requirement model, namely, $M = (V, E, A, C, I)$, to define a software project. Specifically, V denotes the requirement set, where each element v_i represents a requirement; E denotes the dependency relation set, in which each edge ei represents a requirement dependency; A denotes the value set in which each element $a_i(a_i \geq 0)$ represents the benefit corresponding to each requirement v_i; C denotes the resource set in which each element $c_i(c_i \geq 0)$ refers to the resources needed to achieve requirement v_i; and I denotes the heuristic value set in which each element $i_i(i \geq 0)$ represents the heuristic values after

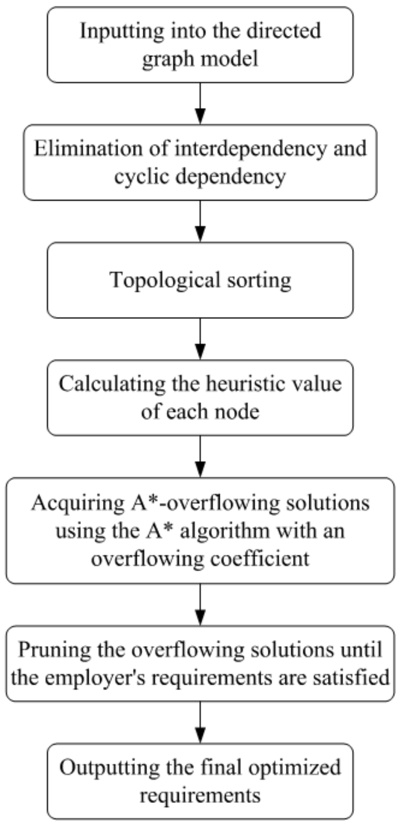

Inputting into the directed graph model
Elimination of interdependency and cyclic dependency
Topological sorting
Calculating the heuristic value of each node
Acquiring A*-overflowing solutions using the A* algorithm with an overflowing coefficient
Pruning the overflowing solutions until the employer's requirements are satisfied
Outputting the final optimized requirements

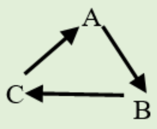
Figure 1. A*-pruning algorithm process

requirement vi has been met. An A*-pruning algorithm is proposed based on the idea of heuristic value in the A* algorithm. First, a directed graph G is constructed to represent the requirements and their dependency relations. This graph is then converted into an AOV network. After assessing the heuristic value for each node, we expand the nodes according to the assessed values. Consequently, the overflowing solutions can be acquired. Finally, these solutions are pruned to meet employer requirements. Fig. 1 illustrates the A*-pruning algorithm procedure. As shown in the figure, the main difference between our algorithm and A* is the parts of overflowing and pruning.

3.2. **Construction of the directed graph.** The directed graph model that describes the dependency relations of the requirements is constructed according to the following rules. For a directed graph that represents a software project, $G = (V, E)$; each vertex v_i represents a requirement and each edge $e = (v_i, v_j)$ denotes a dependency relation; that is, requirement v_i must be met to achieve requirement v_j. In graph theory, this directed graph can be regarded as an AOV network. In such a network, v_i is regarded as the precedent vertex of v_j given $e = (v_i, v_j) \in E$. A directed cycle is generally not allowed in an AOV network. However, such a cycle may be generated because of the complexity of the requirement dependency. This issue should be addressed and is presented in Section III.C. Table II lists the general dependency types and corresponding legends.

3.3. **Elimination of interdependency and cyclic dependency.** In this study, we designed a legalization algorithm for AOV and converted the directed graphs with cyclic dependency and interdependency relations into the equivalent AOV networks. The specific procedure is described as follows. (1) If all the vertices in the graph have been detected,

TABLE 2. Common dependency relations among requirements

Types of dependency relations	Legend for directed graph	Illustration
precedence dependency control	A→B	Requirement B depends on Requirement A.
Interdependence	A↔B	Requirements B and A are interdependent.
cyclic dependency		Requirement A depends on Requirement C; Requirement B depends on Requirement A; Requirement C depends on Requirement B.

then the algorithm is terminated; else, an undetected vertex v is selected and the algorithm proceeds to (2).

(2) The direct precedent vertices of V comprise set P, and the subsequent vertices of V comprise set Q. Then, the algorithm progresses to (3).

(3) If the intersection of P and Q is empty, then the algorithm proceeds to (4); otherwise, it progresses to (6).

(4) Set P' is composed of the precedent vertices of all the vertices in P, and set Q' is composed of the subsequent vertices of all the vertices in Q. If $P = P'$ and $Q = Q'$, then v is labeled as detected. Subsequently, the procedure reverts to (1); otherwise, the algorithm proceeds to (5).

(5) Let P be the union of P and P' and Q the union of Q and Q'. Then, the procedure progresses to (3).

(6) J is referred to as the intersection of P and Q. Vertex $j$vis chosen from J, and vertex v is merged with j to form a new vertex v'. The values of v' (namely, a) are the sums of the values of vertices v and j, and similarly the resource cost (namely, c) are the sums of the corresponding values of vertices v and j, i.e. $a(v') = a(v) + a(j)$ and $c(v') = c(v) + c(j)$.

Furthermore, the sets of the precedent and the subsequent vertices of v' are produced from the union of the corresponding sets v and j. Vertices v and j are deleted along with the related edges, and new edges are added according to the precedent and subsequent vertices of v'. Moreover, v' is labeled as undetected. Then, the procedure returns to (1).

The interdependency and cyclic dependency relations in the directed graph can be converted into a vertex created as a combination of various requirements by following the aforementioned steps. Meanwhile, the original directed graph is simplified to form an AOV network.

3.4. **Topological sorting to calculate the heuristic values of nodes.** In the process of developing a software project, requirements with high value and that consume less resources (i.e., high value-to-cost ratio) should be implemented first to maximize employer benefits given limited resources. However, the achievement of a requirement with high value-to-cost ratio may depend on the meeting of a few requirements with low value-to-cost ratios given the dependencies and constraints among the requirements. Thus, the value-to-cost ratio of the entire system may decrease significantly. A simple local optimization algorithm cannot obtain a satisfactory global optimal solution when designing a requirement optimization algorithm because of the lack of global information. To address

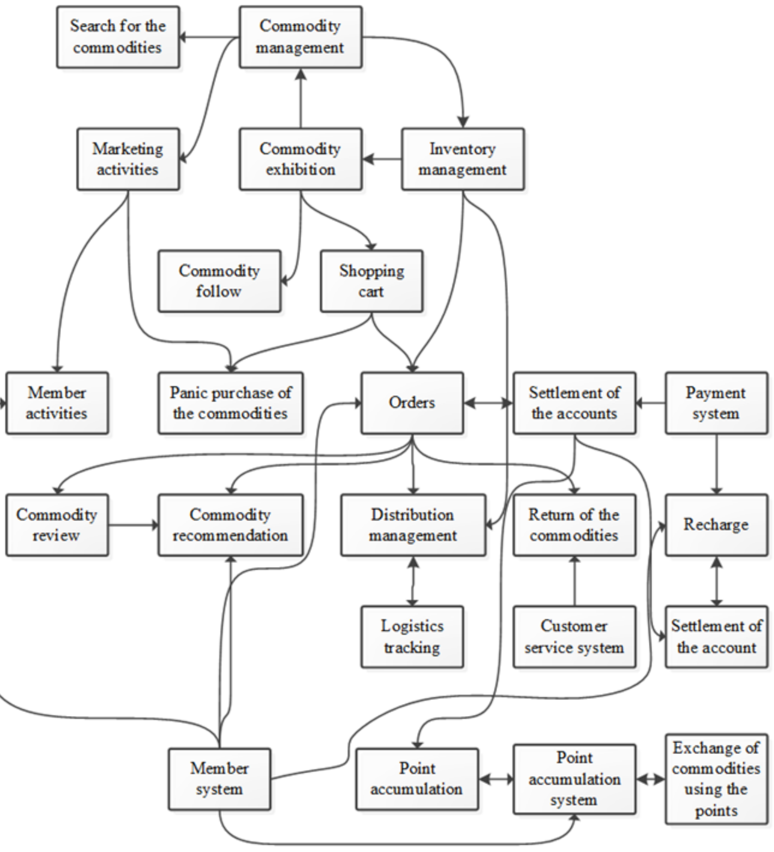


FIGURE 2. Requirements and their dependencies in an E-commerce system

this problem, a heuristic function is introduced into the search algorithm; then, global information is assessed carefully based on the requirement dependency. This process can improve the value-to-cost ratio of an entire system after requirement optimization. The total resources and costs needed to meet user requirements in a software system are generally fixed. To optimize overall value-to-cost ratio, we must consider both the value-to-cost ratio of current requirement examination and potential benefits when travelling along the AOV network to select requirements. In other words, this approach can continuously select subsequent requirements with high value-to-cost ratio as well. The heuristic value i can be defined as follows:

$$i_t = \sum \left(i_j * \frac{c_t}{\sum c_k} \right) + a_t \tag{1}$$

where i_j denotes the heuristic value of v_i, which is the direct subsequent vertex of V_t[i.e., an edge (v_t, v_j) exists in the network] and c_k denotes the resource cost of $v_k[v_k$ is the direct precedent vertex of v_j, i.e., an edge (v_k, v_j) exists in the network]. The definition of heuristic value i is recursive; therefore, topological sorting should be conducted on the AOV network before calculating the heuristic values to obtain the topological sequence of S_T. Then, the heuristic value of each vertex in sequence S_T is calculated in the reverse order; this process avoids the repetitive computation in the recursion procedure. Finally, a directed graph model that considers requirement dependencies is constructed successfully.

3.5. **A* process in the A*-pruning algorithm.** Following the construction of the directed graph model, the requirement optimization problem is equivalent to the problem of effectively searching for a path with a high value-to-cost ratio in a directed graph. Hart et al. proposed the A* search algorithm for a pathfinding problem with given starting and terminal points and proved that the A* algorithm can generate an appropriate heuristic function for the optimal path [31]. The evaluation function in the A* algorithm is written as:

$$f(x) = g(x) + h(x), \tag{2}$$

where $g(x)$ denotes the cost to proceed from the initial state to state x and is thus referred to as the cost function. $h(x)$ denotes the estimated cost required to travel from state x to the optimal state with the algorithm and is thus referred to as the heuristic function. The closer the value of the heuristic function is to the true value, the more quickly the A* algorithm can approach the terminal pathfinding point. The A* algorithm cannot be applied directly because the terminal pathfinding point does not exist in the requirement selection problem. Nonetheless, we can incorporate the idea that the A* algorithm can quickly approach the terminal point by adopting the weighted cost and heuristic functions. The A*-pruning algorithm is proposed to search for paths with a high value-to-cost ratio and to satisfy the dependency relations of the requirements. Specifically, we design a heuristic function that can assess potential value-to-cost ratio and maximize the overall value-to-cost ratio of the system on the premise that each requirement can satisfy its dependency relations under the constraints of limited resources through the overflowing-pruning process. To fit the optimal value-to-cost ratio, the evaluation function $f(v)$ is used to assess the effect of one vertex $v$on the value-to-cost ratio of the entire software project.

$$f(v) = g(v) + h(v) \tag{3}$$

where the cost function $g(v)$is written as:

$$g(v) = \frac{\sum a_i}{\sum c_i} \tag{4}$$

In this equation, v denotes the vertex that may be explored further. a_i and c_i denote the cost and resource requirements of vertex v and of all the vertices explored (i.e., whose requirements have already been met), respectively. The heuristic function $h(v)$ is defined as:

$$h(v) = \theta * i_j \tag{5}$$

where i_j denotes the heuristic value of vertex v_j and θ denotes the empirical parameter that aims to match the heuristic value to the true value. θ value is set to 0.2 in this work, and i can be calculated as follows:

$$i_t = \sum \left(i_j * \frac{c_t}{\sum c_k} \right) + a_t \tag{6}$$

When the heuristic value is obtained, the algorithm runs according to the procedures described below until the terminal condition is satisfied (i.e., employer requirements are met).

The A* process in the A*-pruning algorithm is described as follows:

(1) An initial set of expansible vertices (also called the expansible set) and a set of expanded vertices are inputted. The expansible set cannot be empty.

(2) The estimated values of all vertices in the expansible set are calculated according to the evaluation function in the A*-pruning algorithm.

(3) A vertex is selected from the expansible set with the maximum estimated value and then added to the expanded vertex set.

(4) The variation of the expansible set is computed based on the rules of the AOV network.

(5) When the terminal condition is satisfied or the expansible set becomes empty, the algorithm stops outputting expanded nodes; otherwise, the process reverts to (3).

Once the process is terminated, the resultant optimal solution set is a sub-network of the original AOV network. This set is referred to as the solution of the A* process, is abbreviated as A* solution, and is denoted as G_A. Only one vertex is expanded at each iteration in the A* process, and the time complexity of the operations in each iteration is denoted by $O(n)$. Therefore, the time complexity of the A* process is represented by $O(n^2)$.

3.6. **Overflowing coefficient and pruning process in the A*-pruning algorithm.** The solution obtained for the problem is incomplete; thus, an error is detected between the estimated value and the actual value. The A* solution can then be optimized further toward the global optimum. For example, many vertices with high value-to-price ratios depend on a large number of precedent vertices in an AOV network. When employer resources are limited, these vertices are difficult to expand in the A* process and may limit the improvement of the overall value-to-cost ratio. In addition, certain vertices are challenging to expand because their subsequent vertices are composed of value-to-cost ratio vertices that are both high and low. Thus, a solution can be improved if the A* solution set can be expanded to cover more vertices with high value-to-cost ratio and if solutions are pruned to meet resource constraints.

Specifically, an overflowing coefficient is set ($\alpha > 0$) during the A* process of the proposed A*-pruning algorithm. This coefficient affects the terminal condition of the algorithm and represents the overflowing limit of the solution set. For example, if an employer requires an optimization solution under the condition that the expenditure cannot exceed 50% of total resources given $\alpha = 0.2$, then the maximum expenditure in the A* process should be determined according to the following formula: $(1 + \alpha) \times 50\% = 60\%$. Once the expenditure limit is reached, the algorithm terminates and outputs the corresponding overflowing solution denoted as G_{A+}. α is related to the specific structure of the AOV network; when is too small, the space for local optimization improvement is insufficient and the solution approaches the A* solution. When α is too large, the overflowing solution may deviate from the global optimum under the constraints, and the post-pruning solution may fall into the trap of the local optimum.

The overflowing solution G_{A+} exceeds the resource limit; therefore, the A*-pruning algorithm first selects the vertices without subsequent vertices from G_A. Then, this algorithm removes the vertex with the minimum value-to-cost ratio. This process is repeated until the original requirements of the employer can be met, and the final set of optimal requirements is outputted. The pruning process is described in Algorithm 2.

The *resource_sum* function is used to calculate the total resources consumed by a requirement set, whereas the *delete_worst_v* function is applied to delete the vertex with the minimum value-to-cost ratio and without subsequent requirements in the given requirement set.

Algorithm 2: A*-pruning algorithm (pruning process) During the pruning process, the time complexity of selecting and removing a vertex is denoted by $O(n)$. A maximum of n vertices can be removed. Thus, the overall time complexity is represented by $O(n^2)$.

By combining the time complexities in both the A* and pruning processes, the time complexity of the A*-pruning algorithm is denoted by $O(n^2)$. A requirement optimization solution can therefore be obtained in a polynomial time.

3.7. **Case study.** The requirements of a typical E-commerce system (shown in Fig. 2) are considered in a case study to explain the proposed approach.

An E-commerce system is a typical software system in these years, which should generally be capable of managing commodities and displaying them to consumers online. When an employer chooses one or several commodities, an order is created. Then, the employer

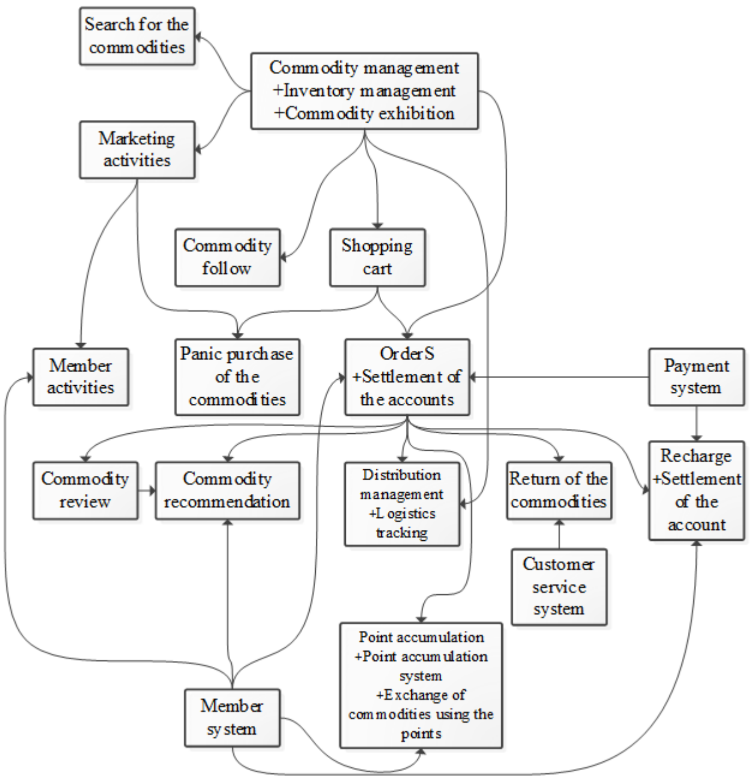

Search for the commodities
Commodity management +Inventory management +Commodity exhibition
Marketing activities
Commodity follow
Shopping cart
Member activities
Panic purchase of the commodities
OrderS +Settlement of the accounts
Payment system
Commodity review
Commodity recommendation
Distribution management +Logistics tracking
Return of the commodities
Recharge +Settlement of the account
Customer service system
Point accumulation +Point accumulation system +Exchange of commodities using the points
Member system

Figure 3. Legal AOV network after processing

is required to fill in the shipping address, and payment options are provided. Once the employer completes the payment process, the merchant should prepare the order and ship the commodities out through logistics. Once the employer receives the requested commodities, the main transaction process is completed. Viral marketing activities and user feedback are also involved in this procedure. There are complex dependencies between the corresponding system modules for these functions, so we choose it as an case of our optimization algorithm. As depicted in Fig. 2, the system includes 24 requirements, 24 direct precedent dependencies, 5 interdependencies, and 1 cyclic dependency. By adopting the algorithm described in Section III.C, cyclic dependency can be eliminated and an AOV network derived, as displayed in Fig. 3. No interdependency or cyclic dependency relations are observed in the network.

As exhibited in Fig. 3, the member system module consumes much development resources but cannot generate high values directly; that is, this module has a low value-to-cost ratio. A few modules rely on the member system, such as member activities, point accumulation, and commodity recommendation, to achieve their functions. On the one hand, the member system module is difficult to expand with local optimum-based algorithms (such as the uniform cost search algorithm) because its value-to-cost ratio is low. This low ratio may result in the abandonment of subsequent requirements with high value-to-cost ratios. On the other hand, a global-optimum-based algorithm is likely to omit the requirements at the boundary (i.e., requirements with low heuristic values and no subsequent dependency). Even algorithms that combine both local and global characteristics (such as the A* process in the proposed algorithm) may terminate in the process of expanding a vertex with a low value-to-cost ratio but a high heuristic value; this termination leads to sub-optimality. To address the aforementioned problems, the A*-pruning algorithm is proposed in which weights are considered for both the local and global information on each requirement. In addition, the A* overflowing coefficient is introduced to increase the optimization search space, and the pruning process is adopted to filter low value-to-cost requirements at the boundary of the overflowing solution set. In the next section, we analyze the A*-pruning algorithm experimentally.

4. **Experimental Analysis.**

4.1. **Experimental data.** To avoid the subjectivity of experimental data, we developed a program for simulating user requirements. This system is based on the requirement architecture model of the E-commerce system and can produce several groups of simulation data (the number of groups is preset). The system also assigns values and resource costs for each requirement in each data group at random. To ensure that the experimental data were statistically stable, we applied 20 groups of simulated requirement data that were randomly generated by this program to test and estimate this algorithm.

4.2. **Evaluation index.** The total values and total resources of all the requirements in the requirement system are fixed for a software project. If all these requirements can be satisfied by the resources of the employer, then all the requirements will certainly be achieved and optimized. Therefore, limit conditions must be set for requirement optimization so that the system retains room for this process. Employers typically expect to obtain the maximum ratio of realized values to input resources. Therefore, the effectiveness of the proposed algorithm is verified by the following two evaluation indices. a) The maximum value can be achieved by the system when the total resources are fixed. In reality, the total available resources of a software project are always limited and cannot meet all requirements. Therefore, maximizing employer benefits is an important index for assessing a requirement optimization algorithm. When the total quantity of resources is fixed, the ratio of the maximum value among various selection schemes can be equivalent to the ratio of the value of the requirement given the resource cost in each scheme. In this study, the value-to-cost ratio of the optimization solution set when total resources are limited can be calculated as:

$$R_{cost\_limit} = \frac{\sum a_i}{cost\_limit} \tag{7}$$

where *cost_limit* denotes the total resources limited by the employer and ai denotes the value of the requirements in the optimization solution set G. Statistically, we utilized the average value of the value-to-cost ratios obtained under different resource limitations to assess the optimization results of the algorithm. b) The minimum resources required for the system to achieve a certain value. Another common issue in the implementation of a software project involves the amount of resources that should be invested to achieve a certain value goal. From the perspective of an employer, the amount of required resources that corresponds to the requirement optimization scheme should be minimized. Therefore, the occupation of a few resources in the optimization scheme increases value-to-cost ratio given the same value goal. This outcome indicates the superiority of the algorithm. In this study, the value-to-cost ratio of the optimization solution set can be calculated as follows when the value goal of the system is fixed:

$$R_{value\_limit} = \frac{value\_limit}{\sum c_i} \tag{8}$$

where *value_limit* denotes the total value that should be achieved according to the requirements of the employer and c_i denotes the resources occupied by the requirements in the optimization solution set G. Statistically, the average value of the value-to-cost ratios under different value goals is adopted to assess the optimization results.

Table 3. Parameters for test data under different conditions

Method	Resource-limited conditions (cost_limit)	Value-preset conditions (value_limit)
A* overflowing coefficient	0.25	0.2
Percentage of limited resources	30%, 50%, 70%	-
Percentage of the value goal	-	30%, 50%, 70%

4.3. **Experiment settings.** According to the typical limit conditions and evaluation indices in software development, the following parameters were set to verify and assess the algorithm. To illustrate the role of the A*-pruning algorithm in improving the overall value-to-cost ratio of the system, another five optimization methods were considered to be baseline methods in the contrast analyses performed on the 20 aforementioned groups of requirement data. These methods were based on breadth-first traversal, depth-first traversal, uniform cost, greedy, and A* algorithms (referring to the A* process in the A*-pruning algorithm, and the same below).

4.3.1. *Breadth-first-based optimization.* Breadth-first traversal is a widely applied graph traversal strategy. The method starts from a specified vertex in the graph and traverses all the neighboring vertex sets of this vertex preferentially. Therefore, breadth-first-based optimization begins at a given vertex (expansible vertex); then, it preferentially traverses all requirement vertexes that are reachable and are adjacent to the given vertex in the AOV network model.

4.3.2. *Depth-first-based optimization.* As another common graph traversal strategy, depth-first traversal starts from a specified vertex, visits a certain neighboring vertex, and finally traverses all other vertices in the neighborhood that has not been visited previously. Depth-first-based optimization begins at a given expansible vertex and then traverses the reachable requirement vertices in the AOV network model.

4.3.3. *Uniform cost-based optimization.* Uniform cost-based optimization is a part of the A* algorithm and does not contain information in which the evaluation function $f(x)$ is equal to $g(x)$. Therefore, this optimization method starts from a given expansible vertex and gradually selects reachable requirement vertices with the maximum value-to-cost ratios in the AOV network model.

4.3.4. *Greedy algorithm-based optimization.* The greedy algorithm is the heuristic part of the A* algorithm. The corresponding evaluation function is expressed as $f(x) = g(x)$. Therefore, greedy algorithm-based optimization starts from a given expansible vertex and gradually selects the reachable requirement vertices according to favorable heuristic values in the AOV network model.

4.3.5. *A* algorithm-based optimization.* The A* algorithm combines the greedy and uniform cost algorithms with the evaluation function $f(x) = g(x) + h(x)$. Therefore, the greedy algorithm-based optimization starts from a given expansible vertex and gradually selects the reachable requirement vertices given favorable assessment values in the AOV network model. The average value-to-cost ratio is introduced into the present experiment; this ratio represents the ratio of the total values of all requirements to total resource cost.

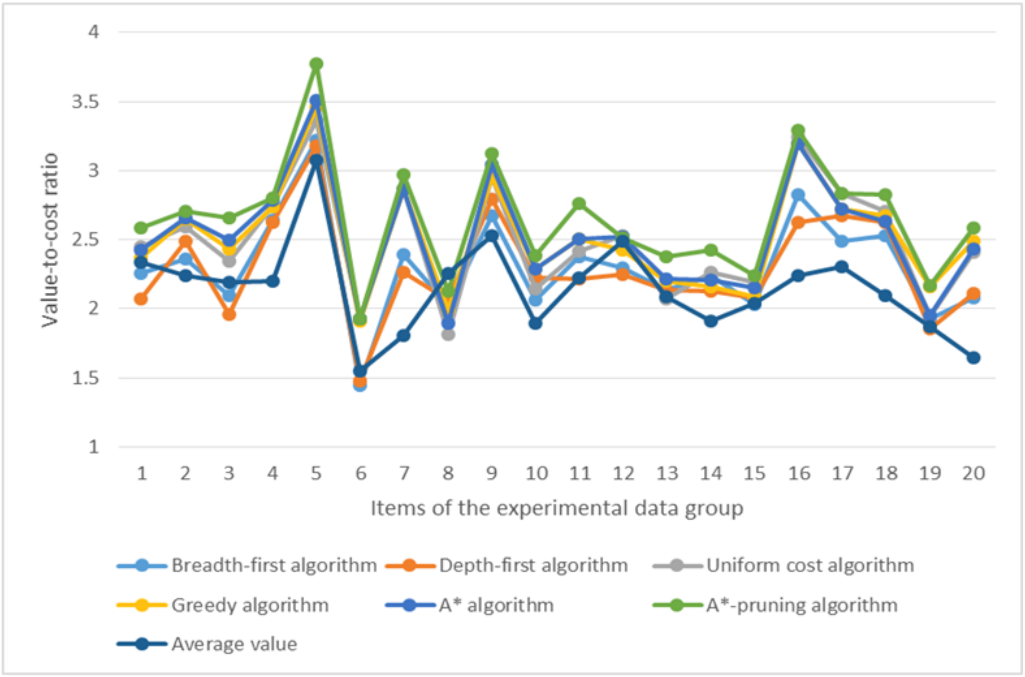


FIGURE 4. Experimental results given limited resources. The x-coordinate denotes the serial number of the experimental data set, and the y-coordinate represents the average value of value-to-cost ratios under the limited resource percentages of 30%, 50%, and 70%.

4.4. **Experimental results.** Denotes the serial number of the experimental data set, and the y-coordinate represents the average value of value-to-cost ratios under the limited resource percentages of 30%, 50%, and 70%.

As shown in Fig. 4, the A*-pruning algorithm reported favorable results in the requirement optimization of 20 groups of random data when the total quantity of resources was limited. The value-to-cost ratio obtained with the A*-pruning algorithm was the most favorable among those generated by other methods for most test groups. In fact, this ratio outclassed the average value-to-cost ratio of the system and the results of the extensively applied breadth-first- and depth-first-based optimization methods. The results produced by the other methods were comparable with those obtained with the A*-pruning algorithm in only a few groups.

Fig. 5 depicts the experimental results under different value goals. The A*-pruning algorithm also produces favorable results under value-preset conditions, as under resource-limited conditions. With these two figure of merit as standards, the A*-pruning algorithm-based optimization scheme can generate a preferable value-to-cost ratio under different limit conditions to meet the goal of requirement optimization. Table IV lists the average value-to-cost ratios obtained with different optimization methods for 20 groups of test data. To analyze the variations of the optimization results further under the limitations of system resources and value goals, a fixed data group was selected to test the value-to-cost ratios of the optimal requirement sets obtained with various algorithms. The resource limitation was gradually increased by setting the stepping interval to 10% of the total quantity of the needed resources. The related experimental results are presented in Fig. 6.

Under different resource limitations, the A*-pruning algorithm reported a favorable optimization result that was either superior or equivalent to the results calculated by other

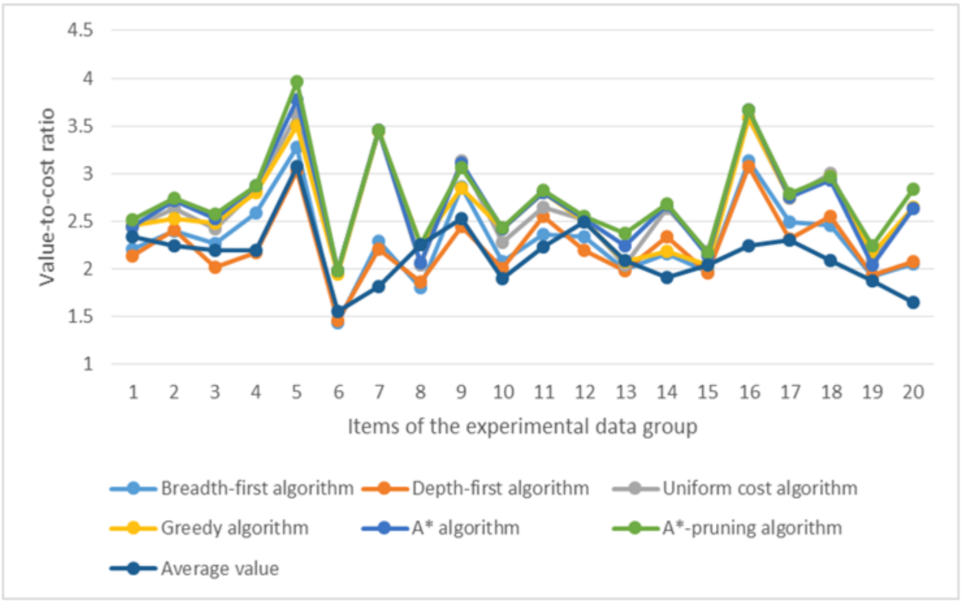


Figure 5. Experimental results under different value goals. The x-coordinate denotes the serial number of the experimental data set, and the y-coordinate represents the average value of value-to-cost ratios when value can be attained to the extent of 30%, 50%, and 70%.

Table 4. Value-to-cost ratios of optimal requirement schemes generated using different optimization methods

Method	Resource-limited condition	Value-preset condition
Breadth-first-based optimization	2.305603	2.302949
Depth-first-based optimization	2.292279	2.236922
Uniform cost-based optimization	2.491817	2.645873
Greedy algorithm-based optimization	2.515532	2.619222
A* algorithm-based optimization	2.522743	2.688632
A*-pruning algorithm-based optimization	2.654278	2.747729
Average value-to-cost ratio	2.149207	2.149207

algorithms. Conclusively, we can obtain stable optimization results using the method proposed in this study. By contrast, the variation tendencies of the results calculated with different algorithms indicate that the more abundant resources are (i.e., the resource limitation is close to the total quantity of the resources needed), the smaller the room for optimization is. These findings also fit well with the actual results of software project development simulation. We also applied 10% of the total values as the stepping interval and tested the value-to-cost ratios of the optimal requirement solution sets established with different algorithms by gradually raising the value goal. The related experimental results are displayed in Fig. 7.

Under the conditions of different value goals, the A*-pruning algorithm can generate favorable requirement optimization results.

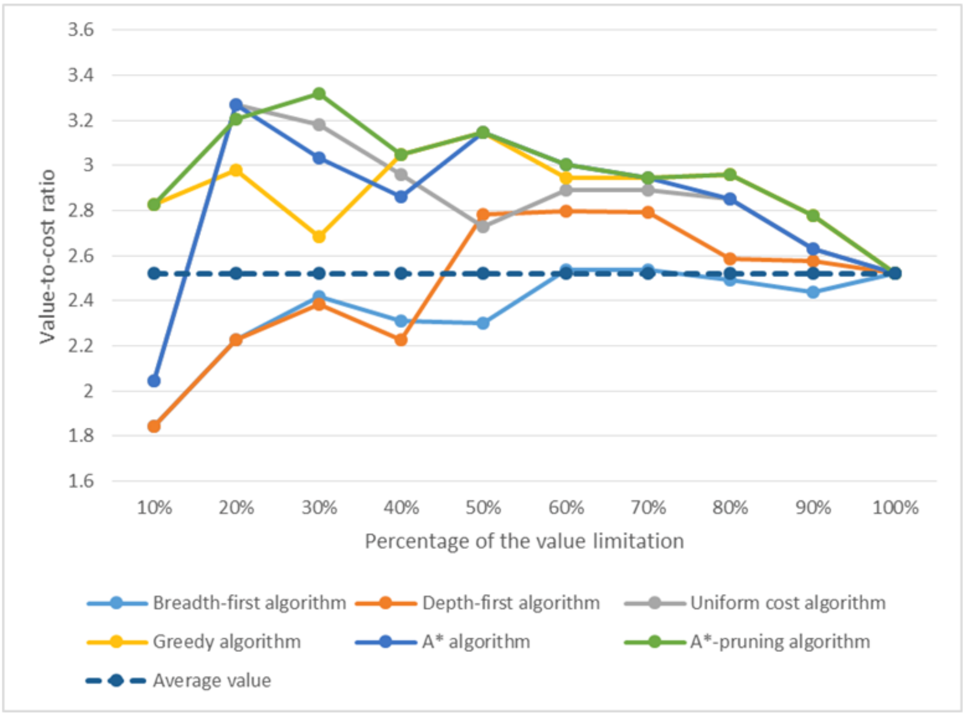


FIGURE 6. Variations in the value-to-cost ratios of the optimal requirement solution sets as generated by different algorithms with varying resource limitations

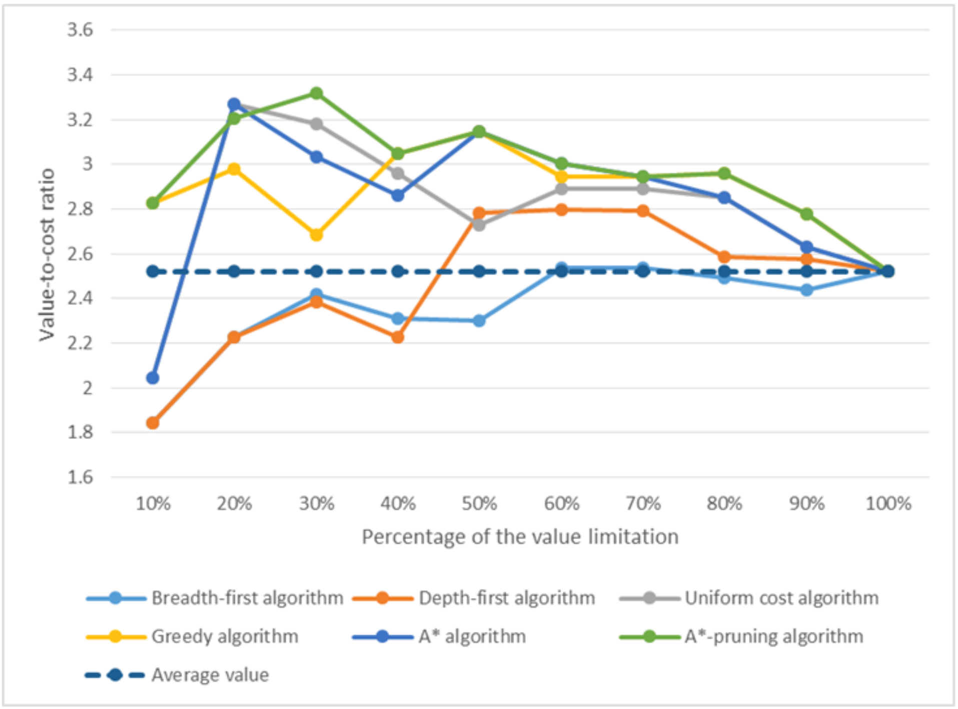


FIGURE 7. Variations in the value-to-cost ratios of optimal requirement solution sets obtained using different algorithms given varying value goals

In summary, when the employer requirements vary, we can obtain optimization results with high value-to-cost ratios using the A*-pruning algorithm on the premise of satisfying the dependency relations of the requirements. As the findings of the experiment conducted under different limitation conditions, the average value-to-cost ratio of the optimal requirement solution set that was calculated using the proposed algorithm was 27% higher than that expected for the system. In fact, the proposed method enhanced the expected value-to-cost ratio of the solution set by 10% more than the other common algorithms did for the optimal requirement solution sets. Consequently, the A*-pruning algorithm is a more favorable requirement optimization method than the other algorithms described in this article.

5. **Conclusion.** In this article, the A*-pruning algorithm is developed by modeling and analyzing the types of dependency relations in a software project based on the applications of topological sorting, the heuristic search method, and the post-pruning method. By using this algorithm, we can obtain an optimization solution set with a high value-to-cost ratio when the dependency relations among requirements are satisfied and when the resource costs of the employer are limited. We present a scientific and reasonable optimization method for requirement analysis in the field of software engineering; nonetheless, optimization is incomplete because of the project properties and because of certain differences between the results derived through heuristic search and the actual results. Experimental results show that our proposed approach can achieve better performance than state-of-the-art baselines.

Acknowledgment. We would like to thank the anonymous reviewers for their valuable and thorough comments. This work is supported by "the 13th Five-Year" National Science and Technology Major Project of China (Grant No. 2017YFC0702204) and the National Natural Science Foundation of China (Grant No.61173021).

REFERENCES

[1] Nuseibeh B, Easterbrook S. Requirements engineering, pp. a roadmap, *Proceedings of the Conference on the Future of Software Engineering. ACM*, pp. 35-46, 2000.

[2] The Standish Group Report: CHAOS. The Standish Group, 1995.

[3] European Software Institute: European User Survey Analysis. Technical Report, Esi-1996-TR95104, 1996.

[4] H, G., Who killed the virtual case file? *IEEE Spectr.*, vol. 42, no. 9, pp. 24-35, 2005.

[5] http://www.ibm.com/developerworks/cn/rational/tracing-project-requirements/index.html

[6] J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The Next Release Problem, *Information and Software Technology*, vol. 43, no. 14, pp. 883890,December 2001.

[7] X. Franch and A. Neil, Maiden. Modelling Component Dependencies to Inform Their Selection, *In Proceedings of the 2nd International Conference on COTS-Based Software Systems (ICCBSS 03),* volume 2580 of LNCS, pages 8191,Ottawa, Canada, 10-12 February 2003. Springer.

[8] D. Greer and G. Ruhe, Software Release Planning: An Evolutionary and Iterative Approach. Information & Software Technology, vol. 46, no. 4, pp. 243–253,March 2004.

[9] Y. Zhang, M. Harman, Search based optimization of requirements interaction management, i*n: Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE 10), IEEE, Benevento, Italy,* 2010.

[10] Yuanyuan Zhang and Mark Harman and Soo Ling Lim Search Based Optimization of Requirements Interaction Management Department of Computer Science, U*niversity College London RN/11/12*, 2011.

[11] Y. Y. Zhang and M. Harman and S. Ling Lim, Empirical Evaluation of Search based Requirements Interaction Management, *Information and Software Technology*, vol. 55, no. 1, pp. 126-152.

[12] J, W.C. Karlsson, Regnell B, An Evaluation of Methods for Prioritizing Software Requirements. Information and Software Technology, 1998.

[13] J. Karlsson, Software requirements prioritizing, *Requirements Engineering ., Proceedings of the Second International Conference on. IEEE,* pp. 110-116, 1996.

[14] N. Mead, Requirements Prioritization Case Study Using AHP[EB/OL].2010.https://buildsecurityin.us-cert.gov/articles/best-practices/requirements-engineering/requirements-prioritization-case-study-using-ahp.

[15] N. Mead An Evaluation of Cost-Benefit Using Security Requirements Prioritization Methods, 2010.

[16] Donald Firesmith: "Prioritizing Requirements," *in Journal of Object Technology*, vol. 3, no. 8, September-October 2004, pp. 35-47.

[17] D. Gree and G. Ruhe , Software release planning: An evolutionary and iterative approach. Inf. Softw. Technol. 46, 4, 243–253, 2004.

[18] K. Praditwong , X. Yao, A new multi-objective evolutionary optimisation algorithm: the two-archive algorithm, *Computational Intelligence and Security, 2006 International Conference on. IEEE*, 1, pp. 286-291, , 2006.

[19] Y. Zhang, M. Harman, Search based optimization of requirements interaction management, *in: Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE '10), IEEE, Benevento, Italy*, 2010.

[20] T. Y. Li, L. Liu, and D. W. Zhao et al., Eliciting relations from requirements text based on dependency analysis, *Chinese Journal of Computers,* vol. 1, pp. 54-62, 2013.

[21] S. T. Luo, C. H. Zhang, Y. Jin et al. Determination of cross-cutting concerns by requirement dependency , *Journal of Jilin University (Engineering and Technology Edition)*, (4), 2011

[22] P. Carlshamre, Release Planning in market-driven software product development: *Provoking an understanding. Requir. Engin.* vol. 7, no. 3, pp. 139–151, 2002.

[23] G. Ruhe, M. O. Saliu , The art and science of software release planning, *IEEE Softw.* vol. 22, no. 6, pp. 47–53.

[24] C. Li, V. D. Akker, ER, S. Brinkkemp, and, G. Dipn, Integrated requirement selection and scheduling for the release planning of a software product. *In Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ'07). Lecture Notes in Computer Science, vol. 4542. Springer*, pp. 93–108.

[25] V. D. M. Akker, S.Bringkkmer , G. Dieplen, and J. Versendaal, Flexible release planning using integer linear programming, *In Proceedings of the 11th International Workshop on Requirements Engineering for Software Quality (RefsQ05).* pp. 247–262, 2005.

[26] J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The Next Release Problem. Information and Software Technology, vol. 43, no. 14, pp. 883–890,December 2001.

[27] D. Greer and G. unther Ruhe, Software Release Planning: An Evolutionary and Iterative Approach. Information & Software Technology, vol. 46, no. 4, pp. 243–253,March 2004.

[28] F. Moisiadis, The fundamentals of prioritizing requirements, in Systems Engineering, Test & Evaluation Conference2002.

[29] V. Laurens, Identifying unwanted requirements dependencies: developing amethod for requirements depend ency modeling, 2012http://essay.utwente.nl/61412/

[30] A. Alebrahim, S. Fa bender, et tel, , Problem-Based Requirements Interaction Analysis, Requirements Engineering: *Foundation for Software Quality Lecture Notes in Computer Science,* vol. 8396, pp 200-215, 2014.

[31] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100-107, 1968.

[32] H. M. Regnell, J. N. Dag, P. Beremark, T. Hjelm, An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software, *Requirements Engineering,* 2001.

[33] L. K. Chan, M. L. Wu Quality function deployment: A literature review, *European Journal of Operational Research*, vol. 143, no. 3, pp. 463-497, 2002.

[34] W. D. Leffingwell, Managing Software Requirements – *A Unified Approach.* 2000.