2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science

Top-k Query for Weighted Interactive Product Configuration

Baijun Chen
Software Engineering
Tongji University
Shanghai, China
1641462@tongji.edu.cn

Tao Feng
Software Engineering
Tongji University
Shanghai, China
1641492@tongji.edu.cn

***Abstract*—Interactive product configuration should be complete and backtrack free. it requires that the system responds quickly. A popular solution to implement interactive product configuration is Binary Decision Diagrams(*BDDs*) which represents the solution space of product model, the structure of *BDDs* is very compact, and the interactive configuration can be carried out in linearly dependent time of *BDD's* size. In this paper, we proposed the concept of *top-k query* about weighted interactive product configuration based on *BDDs*. the solution space of *BDD* may be exponential explosion in practice, we defined weight value for each valid configuration, the weight represents price, quality etc., in the interactive phase, users input a rule and *k*, the system lists top-k maximum(minimum) valid configurations to user for reference. Using our code, we solved a number of problems including those from real application and synthetic data set. Our experiment results showed that our algorithm had a faster response time than the traditional algorithm.**

***Keywords*-Interactive product configuration Binary decision diagrams Top-k query.**

I. Introduction

Interactive product configuration is often presented as an application in constraint satisfaction problem(*CSP*) [1]. It requires that the system has the features of completion, free backtrack and quick response. The product model consists of a set of product variables with finite domains and a set of product rules. These rules define the relationship between variables. A valid product configuration is an assignment to each variable that satisfies all the product rules. Users can choose freely among any valid configurations and will be prevented from making choices that no valid configuration satisfies. In each iteration of the algorithm, invalid values in the domain of the variables are pruned away quickly, the user then assigns value to unassigned variable and the algorithm terminates until each variable has only one assignment.

CSP had been widely studied. Some extended frameworks had been studied. Freuder and Wallace [2] considered its special case called *maximal constraint satisfaction problem(Max-CSP)*, in which all constraints have the same weight; they extended some basic backtracking and local consistency methods for *CSP* to *Max-CSP*. Wallace and Freuder [3] investigated some heuristic methods for *Max-CSP*, which were based on the min-conflicts heuristic. Wallace [4] said the min-conflicts heuristic combined with the random walk strategy was a most successful method for *Max-CSP*. Nonobe and Ibaraki [5] considered constraints with different weights, given a number of constrains and their weights of importance, which asks to minimize the total weight of unsatisfied constraints, they proposed an improved tabu search algorithm for *WCSP*. Jason Li [17] solved CSPs in parallel, they proved that constraint satisfaction problems without the ability to count are solvable by the local consistency checking algorithm. Kuang J [18] proposed interactive product configuration driven by customer requirements priority, AC-3 algorithm was adopted to remove incompatible elements in variable domain, but they didn't consider the response time of the system. The algorithms are not fit for the our situation, the heuristic has a probability of getting into the local optimality, and they can only find one optimal configuration without considering the running time of the algorithm. Their methods are time-consuming and unsuitable. In the interactive system, the response time is very important. In [19], the author proved that method based on *BDD* are suitable for interactive system, but they didn't consider the different element's importances in each variable's domain, and also the solution space are huge in some scene.

In this paper, we considered a special case of *WCSP*, because elements in the variable domain are different. Elements have different importance to represent price, stability etc.. A valid configuration consists of all variables, each variable has a weight value according to its assignment. Our *WCSP* sum all the weight value of each variable to represent the weight of the valid configuration. The solution space(all valid configurations) which satisfy the user's customization may be exponential explosion, so we consider the *top-k query* concept into our *WCSP*. The system shows the top-k valid configurations with maximum(minimum) weight to the user which helps the user to make better choice. it is also necessary to reduce the response time of the system in the interactive process.

In this paper we developed an efficient *WCSP* solver with *top-k query*. The main advantage of our approach is that we can respond the user with top-k valid configurations with maximum weight quickly without knowing all valid configurations in solution space. Our contributions mainly include the followings:

- The paper implemented a traditional top-k query algorithm for *weighted interactive product configuration*.
- The paper proposed two pruning algorithms(TP and AO) for *weighted interactive product configuration* .

978-1-5386-7445-1/18/$31.00 ©2018 IEEE
DOI 10.1109/DCABES.2018.00090

T-shirt	variable \ index	0	1	2	3	
type	color	Black:0.21	White:0.65	Red:0.12	Blue:0.76	x_1^1, x_1^0
	size	Small:0.33	Medium:0.45	Large:0.12		x_2^1, x_2^0
	print	MIB:0.32	STW:0.22			x_3^0
rule	(print==MIB)>>(color==Black);					
	(print==STW)>>(size!=Small);					

Figure 1. T-shirt problem configure profile(cp).

1, {Black, Small, MIB}
2, {Black, Medium, MIB}
3, {Black, Medium, STW}
4, {Black, Large, MIB}
5, {Black, Large, STW}
6, {White, Medium, STW}
7, {White, Large, STW}
8, {Red, Medium, STW}
9, {Blue, Medium, STW}
10, {Red, Large, STW}
11, {Blue, Large, STW}

Figure 2. Solution space of T-shirt problem.

The rest of this paper is organized as follows. In Section 2, we define *interactive product configuration* solvers based on *BDD*. In Section 3, we introduce our *WCSP* solver. In Section 4, we propose our top-k query algorithm based on *BDD*, this paper proposes two algorithm(two-pointer-based and array-offset based) to prune *BDD* for reducing system response time. In Section 5, We experiment our *WCSP* with real and synthetic data set, the experimental results show the efficiency of our algorithm. In Section 6. we close this paper with a conclusion.

II. Interactive Product Configuration

A. Product Configuration

Interactive product configuration is a special application of constraint satisfaction problems(CSP) where a user is assisted in interactively assigning values to variables by a software tool. This software, called a configurator which assists the user by calculating and displaying the available and valid choices for each unassigned variable. There are two main points in the interactive product configuration. First, the user can freely choose any valid assignment. Second, feedback should be fast enough to allow real-time interaction.

Definition 1. A configuration problem C is a triple *(X, D, F)*, where X is a set of variables x_1, x_2, ... ,x_n. D is the cartesian product of their finite domains. $D = D_1 \times D_2 \ldots D_n$, F=f_1, f_2, ..., f_m is a set of propositional formulas over atomic propositions $x_i = v$, where v ϵ D_i, specifying conditions that the variable assignments must be satisfied. Each f_i is a propositional expression inductively defined by

$$\phi \equiv x_i = v \mid \phi \wedge \psi \mid \phi \vee \psi \mid \rightharpoonup \phi \qquad \text{for each } v \in D_i$$

For logical implication $\phi \Rightarrow \psi \equiv \rightharpoonup \phi \vee \psi$.

Example 1. Consider the T-shirt problem(Fig.1) with three variables(*color*, *size*, *print*), and two rules need to be met, one is *(print==MIB)* $\Rightarrow$ *(color==Black)* which means if we choose *MIB* for *print* then *color==Black* must be chosen, another is *(print==STW)* $\Rightarrow$*(size!=Small)* which means if we choose *STW* for print then *size==small* should't be chosen.

The configure problem C of T-shirt {*X, D, F*}, X={x_1, x_2, x_3} representing *color*, *size*, *print*. Variable domain D_1={*Black, White, Red, Blue*}, D_2={*Small, Medium, Large*}, D_3={*MIB, STW*}. F={F_1, F_2}, F_1={*(print==MIB)*$\Rightarrow$*(color ==Black)*}, F_2={ *(print==STW)*$\Rightarrow$*(size!=Small)*}. So there are totally $|D_1||D_2||D_3|$=24 possibility. There are 11 valid configurations satisfying F_1 and F_2, they form the solution space(Fig.2).

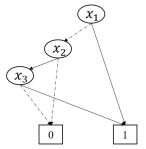


Figure 3. BDD of $f = X_1 + X_2 \cdot X_3$.

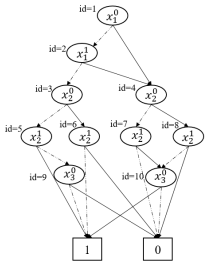


Figure 4. BDD of T-shirt problem.

B. User Interaction

Configurator assists the user interactively to reach a valid product configuration(each variable has one valid assignment), so a faster response speed is necessary. The user backtrack-free assigns to variables as long as the user input one rule. At each step of the interaction, the configurator reports the valid domains to the user based on the current partial assignment resulting from his earlier choices.

C. Binary Decision Diagrams

In computer science, a binary decision diagram(*BDD*) is a data structure which is used to represent a Boolean function. *BDDs* can be considered as a compressed representation of sets or relations. An example of BDD represents $f = X_1 + X_2 \cdot X_3$ is shown in Fig.3, the dotted line indicates that the variable is assigned to 0, the solid line indicates that the variable is assigned to 1.

D. Compiling the Configuration Model for T-shirt Problem

The *BDD* which represents the solution space of the T-shirt problem is shown in Fig.4. In the T-shirt problem, there are three variables: X_1, X_2 and X_3, whose domain sizes are four, three and two. each variable is represented by a vector of Boolean variables. the Boolean vector for the variable X_i with domain D is {$X_i^{l_i-1}$, ..., X_i^1, X_i^0}, where $l_i = \lceil lg|D_i| \rceil$, For T-shirt problem, variable X_2 which

corresponds to the size of the T-shirt, size is represented by the Boolean vector (X_2^1, X_2^0), $\langle X_2^1{:}0, X_2^0{:}0 \rangle$==*Small*, $\langle X_2^1{:}0, X_2^0{:}1 \rangle$==*Medium*, $\langle X_2^1{:}1, X_2^0{:}0 \rangle$==*Large*, any path from the root vertex to the terminal vertex 1, corresponds to one or more valid configurations. For example, the path from the root vertex to the terminal vertex 1, with all the variables taking low values represents the valid configuration *(black,small,MIB)*. Another path with X_1^1, X_1^0, X_2^1 taking low values, and X_2^0 taking high value represents two valid configurations: *(black,medium,MIB)* and *(black,medium,STW)*, in this path the variable X_3^0 is a 'don't care' variable and hence can take both low and high value, which leads to two valid configurations. Any path from the root vertex to the terminal vertex 0 corresponds to invalid configurations.

III. Weighted Constraint Satisfaction Problem

This paper considered weight on the element in each variable's domain. Different elements may have different importance which corresponds to price, quality etc.. the weight of valid configuration is the sum of all its variables. For T-shirt problem, weight information is setted by manufacturer, and is shown in Fig.1, each valid configuration has a weight value(eg. *weight(Black,Medium,MIB)=0.21+0.45+0.32=0.98*). Usually, for configure problem *C*, the number of valid configurations is exponential explosion. In the user interactive phase, the user inputs a rule and top k number, the configurator will respond to the user top-k maximum valid configurations as soon as possible to guide users to make better choices according to weight information. Our research majors in how to improve the response speed of the system for *top-k query* no matter what rules the user inputs.

IV. Top-k Query Algorithm for *WCSP*

A. Traditional Algorithm

When the user assign a new variable and top-k value, the system gets a new *BDD* according to the new assignment, then the traditional algorithm gets all valid configurations based on depth searched on *BDD*. Every time it gets one valid configuration, then calculates its weight value. The min-heap whose size equals k is used to save the finally top-k query results. The running time of the algorithm is shown as follows.

$$T(n)=O(n+h*s*2) \tag{1}$$

n: vertex num of *BDD*; *h*: encode bit num; *s*:num of valid configurations(*n*:10, *h*=5, *s*=11 in T-shirt problem).

Example 2. The *BDD* of T-shirt problem is shown in Fig.4, the weight information is shown in Fig.1. we take a depth first search for *BDD*. then we get all valid configurations, for one valid configuration, such as *array*=$\{x_1^0{:}0, x_1^1{:}0, x_2^0{:}0, x_2^1{:}0, x_3^0{:}0\}$. We init *weight*=0, $\{x_1^0{:}0, x_1^1{:}0\}$, *color=black*, then *weight*+=0.21; $\{x_2^0{:}0, x_2^1{:}0\}$, *size=Small*, then *weight*+=0.45; $\{x_3^0{:}0\}$, *print*=MIB, then *weight*+=0.32; finally, *weight*=0.86. we compare all of them, we can get the finally top-k query results in min-heap.

The disadvantage of this algorithm is that all valid configurations are visited, which means all vertexes in *BDD* are visited, and computing weight is redundant, supposing we get *array1*=$\{x_1^0{:}0, x_1^1{:}0, x_2^0{:}0, x_2^1{:}0, x_3^0{:}0\}$, *array2*=$\{x_1^0{:}0, x_1^1{:}0, x_2^0{:}0, x_2^1{:}0, x_3^0{:}1\}$, for above two valid configurations, we all init the *weight*=0; we must calculate *color*=black, *size=Small* twice, so the computing is inefficiency.

B. Top-k Query Pruning Algorithm Based Two-Pointer(TP)

TP improves the disadvantage of traditional algorithm. TP adopts the pruning algorithm to avoid visiting all vertexes in the *BDD*, so only partial valid configurations are known. It is based on the idea of top-k search, for each valid configuration, the *weight* is the SUM of all its variables, *weight*=$f(v_0,v_1,...,v_n)$, for variable v_i, if we know its assignment, we use its true weight instead v_i, else upper weight instead v_i. Each time when a vertex is visited, TP pre-calculates the upper weight, if the upper weight less than the weight of min-heap top, subsequent vertexes in the current path don't need continue to visit.

TP updates *weight* with two pointers, one pointer points to the current vertex, the other points to father vertex, if they point to the same variable in cp, it indicates that the assignment of the variable is not determined, else it indicates the variable of father pointer points is determined, then TP updates the current *weight*. The running time of the algorithm is shown as follows.

$$T(n)=O(k_1 \times n+h\times s\times k_2),\ 0< k_1 <1,\ 0< k_2 <1, \tag{2}$$

The following concepts are important for TP, TP is shown in algorithm 1,

- *matrix_weight[][]*: weight information of variable domain in cp;
- *leveltocpvar[]*: mapping of variable's level in *BDD* to variables in cp;
- *max_weight[]*: the max weight of each variable in cp file.
- *upper_weight[]*: the sum of unknown variable's max weight.

$$upper\_weight[i] = \sum_{k=i}^{n} max\_weight[k] \tag{3}$$

Example 3. In T-shrit.

- *matrix_weight[][]*={ *[0.21(*Black*), 0.65(*White*), 0.12(*Red*), 0.76(*Blue*)], [0.33(*Small*), 0.45(*Medium*), 0.12(*Large*)], [0.32(*MIB*), 0.22(*STW*)]*},
- *leveltocpvar[]*={*1, 1, 2, 2, 3* }, in which *leveltocpvar[0]*(x_1^0) and *leveltocpvar[1]*(x_1^1) both point the first variable *color* in cp, *leveltocpvar[2]*(x_2^0) and *leveltocpvar[3]*(x_2^1) point second variable *size*, *leveltocpvar[4]*(x_3^0) points third variable *print*.
- *max_weight[]*={*0.76, 0.45,0.32*}, the max weight value of each variable in cp.

- *upper_weight[]={1.53, 0.77, 0.32}*, *upper_weight[0]=0.76+0.45+0.32*, *upper_weight[1]=0.45+0.32*, *upper_weight[2]=0.32*.

Algorithm 1: TP_Query-top-k

Require: *i*:int, *v*:vertex, *x*:*array*$[0,1,...,n]$ of int, *weight*:double
Ensure: top-k max-weight valid configurations
1: **if** *v*.value==0 **then**
2: return;
3: **end if**
4: *weight=getCurrentWeight*($father$, *i*, *weight*,*x*);
5: *upperweight= weight +* upper_weight[leveltocpvar[i]]
6: **if** *upperweight < minheap.top.value* **then**
7: return;
8: **else if** *i*==*n* && *v*.value==1 **then**
9: Add *weight* and *array*$[0,1,...,n]$ to *minheap*, return;
10: **else**
11: **if** *v*.index > i **then**
12: *x*[i]=0;$father$=*i*;TP_Query-top-k(*i*+1,*v*,*x*,*weight*);
13: *x*[i]=1;$father$=*i*;TP_Query-top-k(*i*+1,*v*,*x*,*weight*);
14: **else**
15: *x*[i]=0;$father$=*i*;TP_Query-top-k(*i*+1,*v*.low,*x*,*weight*);
16: *x*[i]=1;$father$=*i*;TP_Query-top-k(*i*+1,*v*.high,*x*,*weight*);
17: **end if**
18: **end if**

Example 4. The *BDD* is shown in Fig.4. Depth first traversal of *BDD*. Init the *weight*=0, the first visited vertex is id=1, then id=2, the current vertex is id=2, father vertex is id=1, they point same variable (*color*), we only know x_1^0=*0*, so the assignment of color has not been known, then the vertex id=3 visited, father vertex is id=2, they belong to different variable in cp, so the assignment of color is known, *color=black*, update *weight*+=0.21. upper weight is *upper(weight)=weight+upper_weight[1]*, if current valid configurations number less then top-k, pruning is prohibited. else if *upper(weight)* less then the top-element of min-heap, then return.

C. Top-k Query Pruning Algorithm Based Array-Offset(AO)

The overall idea is the same as in TP, but pruning ability is better than TP. TP knows partial assignment of levels, it does't know the assignment of variable in cp(only knowing x_0^0=*1*, *color* is still unassigned), so TP use max weight in *color* instead, so the upper bound weight is higher. AO needn't to estimate the upper weight, we set the initial *weight=upper_weight[0]*, in the process of traversing the BDD, update the weight in real time. The running time of the algorithm is shown as follows.

$$T(n)=O(k_1 \times n+h\times s\times k_2),\ 0< k_1 <1,\ 0< k_2 <1, \quad (4)$$

in which k_2 is less then that in TP. AO reorder elements in variable domain, for each variable, we reorder elements according to the weight value from big to small. Before {*Black:0.21, White:0.65, Red:0.12, Blue:0.76*}, after {*Blue:0.76, White:0.65, Black:0.21, Red:0.12,*}.

The following concepts are introduced, the AO is shown in Algorithm 2.

- *offset*{}, mapping from level to *matrix_weight[][]*.
- *position[]*, the current variable assignment in cp.
- init *weight*=$\sum_{k=1}^{n}$ *max_weight*[*k*].

Example 5. In T-shrit problem.

- After reordering, *matrix_weight[][]*={ *[Blue:0.76*{x_1^1=*0*, x_1^0=*0*}, *White:0.65*{x_1^1=*0*, x_1^0=*1*}, *Black:0.21*{x_1^1=*1*, x_1^0=*0*}, *Red:0.12)*{x_1^1=*1*, x_1^0=*1*}], *[Medium:0.45*{x_2^1=*0*, x_2^0=*0*}, *Small:0.33*{x_2^1=*0*, x_2^0=*1*}, *Large:0.12*{x_2^1=*1*, x_2^0=*0*}], *[MIB:0.32*{x_3^0=*0*}, *STW:0.22*{x_3^0=*1*}]} (note: the *BDD* structure also change, and *BDD* is not shown here),
- init the *position[]*={*0, 0, 0*}, so init the *color=Blue*, *size=Medium*, *print=MIB*; *position[0]*∈*[0, 3]*, *position[1]*∈*[0, 2]*, *position[2]*∈*[0, 1]*.
- *offset*{}={x_1^0*:1*, x_1^1*:2*, x_2^0*:1*, x_2^1*:2*, x_3^0*:1*}, *BDD* level is converted to Decimal digits representation according to variable in cp.

Example 6. In T-shrit problem. AO init the *weight=upper_weight[0](0.76+0.45+0.32=1.53)* and *position[]*={*0, 0, 0*}, then if x_i^j takes zero, AO do nothing, else, AP update *weight* and *position*, we give a concrete example, if x_1^0=0, nothing to do, next, we get x_1^1=1, update *weight*=1.53-0.76+0.21=0.98 and position[]={2,0,0}, next x_2^0=0, nothing to do, x_2^1=0, nothing to do, last, we get x_3^0=1,update *weight*=0.98-0.32+0.22=0.88 and *position[]*={*2,0,1*}, finally, AP get {*Black,Medium,STW*}, *weight*=0.88

In conclusion, the traditional algorithm visits all solutions in the solution space, so it is very time consuming; the other two pruning algorithm only visit partial solutions in the solution space, so it reduces system response time; the AO is better than TP.

V. Experiment

In our experiment, we experimented with real data and synthesized data set. One real data set from SA-IC motor, the other two(bike-2, pc-2) real data set are from https://www.itu.dk/research/cla/externals/clib/(CLib : Configuration Benchmarks Library); then we synthesized one data set(syn-data). There are not many studies based on our scenario, so we experimentally compared the traditional algorithm with our algorithms. The description of the experiment data set is shown in table1(*n*: vertex

Algorithm 2: AO_Query-top-k

Require: i:int, v:vertex, x:$array[0,1,...,n]$ of int, $weight$:double, $position$:$array[0,1,...,m]$ of int
Ensure: top-k max-weight valid configurations

```
1:  if v.value==0 then
2:     return;
3:  end if
4:  if weight < minheap.top.value then
5:     return;
6:  else if i==n && v.value==1 then
7:     Add weight and array[0,1,...,n] to minheap,
       return;
8:  else
9:     if v.index > i then
10:       x[i]=0;
11:       AO_Query-top-k(i+1,v,x,weight,position);
12:       x[i]=1;
13:       old_value = the original weight
14:       new_value = new weight according to offset[i]
15:       weight=weight-old_value+new_value;
16:       position[leveltocpvar[i]] += offset[i];
17:       AO_Query-top-k(i+1,v,x,weight,position);
18:       position[leveltocpvar[i]] -= offset[i];
19:    else
20:       {others are the same as above}
21:       x[i]=0;
22:       AO_Query-top-k(i+1,v.low,x,weight,position);
23:       x[i]=1;
24:       AO_Query-top-k(i+1,v.high,x,weight,position);
25:    end if
26: end if
```

Table I
INFORMATION OF EXPERIMENT DATA SET.

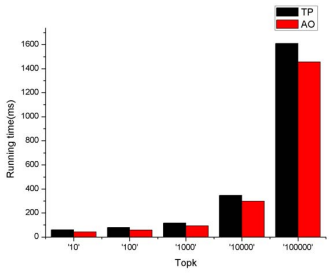
Information	SAIC	Bike-2	Pc-2	Syn-data
n	125126	3129	13332	840
h	279	127	84	86
s	14889012	122461056	1066310	507142944

num of BDD; h: encode bit num; s:num of valid configurations). (CPU:Intel(R) Xeon(R)CPU E5-2620 v3@ 2.40GHz, MemTotal:64G).

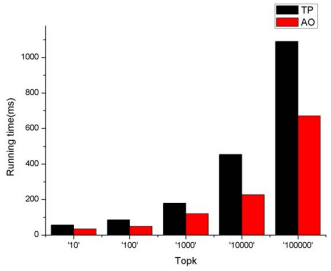
For weight information, we randomly assigned the elements of the variable domain in cp. For distributed parallelism top-k query, we used one master and four slaves. Fig.5 shows the results of our experiment(the running time of traditional algorithm is too long, so it isn't displayed in Fig.5, but it is explained in the description).

VI. CONCLUSION

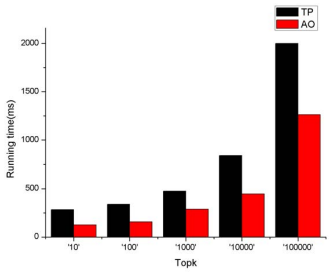
This paper introduced *top-k query* for weighted interactive product configuration, because of the size of solution space is exponentially exploded, we introduced the concept of *top-k query* based on *BDD*, which guides the user to make better choices. Two pruning algorithms(*TP* and *AO*) were introduced in this paper to speed up



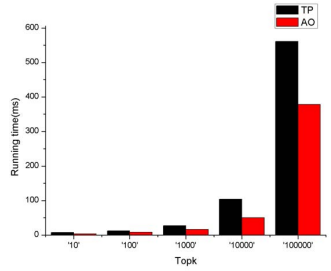
(a)



(b)



(c)



(d)

Figure 5. Running time for different top-k value (a)SAIC (traditional:65s) (b)PC-2 (traditional:2s) (c)Bike-2 (traditional:183s) (d)Syn-data (traditional:626s)

response time, Further research includes implementation and evaluation of *top-k query* by multi-thread.

References

[1] Sabin D, Weigel R. Product Configuration Frameworks-A Survey[J]. IEEE Intelligent Systems & Their Applications, 1998, 13(4):42-49.

[2] Freuder E C, Wallace R J. Partial Constraint Satisfaction[J]. Artificial Intelligence, 1992, 58(13):21-70.

[3] Wallace R J, Freuder E C. Heuristic Methods for Over-Constrained Constraint Satisfaction Problems[C]// Over-Constrained Systems. Springer-Verlag, 1996:207-216.

[4] Wallace R J. Analysis of heuristic methods for partial constraint satisfaction problems[C]// International Conference on Principles and Practice of Constraint Programming. 1996:482–496.

[5] Nonobe K, Ibaraki T. An Improved Tabu Search Method For The Weighted Constraint Satisfaction Problem[J]. Infor Information Systems & Operational Research, 2001, 39(2):131-151.

[6] Hadzic T, Jensen R M, Andersen H R. Calculating Valid Domains for BDD-Based Interactive Configuration[J]. Eprint Arxiv, 2007.

[7] Madsen J N. Methods for interactive constraint satisfaction[J]. 2003.

[8] Hadzic T, Subbarayan S, Jensen R M, et al. Fast backtrack-free product configuration using a precompiled solution space representation[C]// Peto Conference. 2004:131–138.

[9] Jensen R M. CLab: A C++ Library for Fast Backtrack-Free Interactive Product Configuration[M]// Principles and Practice of Constraint Programming CP 2004. Springer Berlin Heidelberg, 2004:816-816.

[10] Mller J, Reif H, Hulgaard A H. Product Configuration over the Internet[C]// Informs Conference on Information Systems and Technology. 2001:33-47.

[11] Subbarayan S, Jensen R M, Hadzic T, et al. Comparing Two Implementations of a Complete[J]. Cp04 Cspia Workshop, 2004:97–111.

[12] Pan G, Vardi M Y. Symbolic Techniques in Satisfiability Solving[J]. Journal of Automated Reasoning, 2005, 35(1-3):25-50.

[13] Weigel R, Faltings B. Compiling constraint satisfaction problems[J]. Artificial Intelligence, 1999, 115(2):257-287.

[14] Subbarayan S. Integrating CSP Decomposition Techniques and BDDs for Compiling Configuration Problems[M]// Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer Berlin Heidelberg, 2005:351-365.

[15] Duffy K R, Bordenave C, Leith D J. Decentralized Constraint Satisfaction[J]. IEEE/ACM Transactions on Networking, 2013, 21(4):1298-1308.

[16] Barto L, Kozik M. Constraint Satisfaction Problems Solvable by Local Consistency Methods[M]. ACM, 2014.

[17] Li J, O'Donnell R. Bounding Laconic Proof Systems by Solving CSPs in Parallel[C]// ACM Symposium on Parallelism in Algorithms and Architectures. ACM, 2017:95-100.

[18] Kuang J, Jiang P. Interactive product configuration driven by customer requirements priority[C]// International Conference on Computer Supported Cooperative Work in Design. IEEE, 2008:58-63.

[19] Subbarayan S. Integrating CSP Decomposition Techniques and BDDs for Compiling Configuration Problems[M]// Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer Berlin Heidelberg, 2005:351-365.

[20] Lu H, Yue T, Ali S, et al. Zen-CC: An Automated and Incremental Conformance Checking Solution to Support Interactive Product Configuration[C]// IEEE, International Symposium on Software Reliability Engineering. IEEE Computer Society, 2014:13-22.