Ming-Yu Shang, Li-Xi Zhang, Jiao-Jiao Zhang, Cheng Wang and Zhan-Shan Li*

**An Optimized MaxRPC Algorithm with a New Search Strategy**

**Abstract:** Max Restricted Path Consistency (maxRPC) is a kind of local consistency for binary constraints in the field. It can achieve noticeably better pruning than arc consistency. Many implementations of maxRPC have been raised in [1]. One of these algorithms (named maxRPC3) has the best time complexity theoretically. However it still has overheads and redundancies for their repeatedly performing constraint checks to some extent. Considering that the process of searching for PC-witnesses can be performed many times and the most constraint checks are performed in this process, the key of improving the performance of maxRPC algorithms is to accelerate this process. In this paper, two algorithms maxRPC3-Alter and maxRPC3 Alter with a new search strategy based on bitwise operation to accelerate the computations of this process are proposed. Experimental results demonstrate that both new algorithms and their light versions have good performance and outperform the best ones among existing algorithms by a large margin, which also constitute a more than viable alternative to arc consistency.

**Keywords:** maxRPC, Constraint Satisfaction Problem, Search Strategy

1 **Introduction**

Constraint Satisfaction Problem (CSP) is the core technologies in the field of artificial intelligence and is one of the most challenging research directions. At present, it has been widely used in various fields of artificial intelligence, such as qualitative reasoning, model-based diagnosis, scene analysis, task scheduling, system configuration, natural language understanding, and scientific experiment planning and so on. Filtering techniques are always used to remove some local inconsistencies in the search algorithms when used in a preprocessing step or during the search. It com-
presses the variable field of the constraint network without changing the constraint graph by adding new constraints or deleting tuples that do not satisfy the constraints. maxRPC is one of the most important filtering techniques for the two constraint networks [1]. Computational experiments show that maxRPC provides a great compromise between the computational cost and the pruning efficiency [2]. MaxRPC removes both the values that have no PC-supports and AC-supports, which is stricter than AC. MaxRPC has been implemented by many different methods as the most important part of constraint programming, such as: maxRPC1 [1], maxRPC2 [3], maxRPC3 [4], maxRPCm [4], and maxRPC3m [5]. According to [5] we know that maxRPC3, lmaxRPC3 and lmaxRPC3m are the most efficient algorithms when used stand-alone among existing algorithms, and lmaxRPC3m has the best performance when solving the instances of the CSP.

In this paper, we propose a new method to improve the performance of maxRPC3 algorithm. Traditional algorithms tend to test each value of $x_k$ in $D(x_k)$ one by one when searching for PC-witnesses, as a result of which, some redundant tuples could be repeatedly checked, which will result in a reduction in the efficiency of the algorithm. The new algorithm that we proposed combines maxRPC3/maxRPC3m algorithms with a more efficient search strategy which exploits special data structures and the bitwise operations during the process of PC-witness searching.

The rest of this paper we talk about these: in the next section, we recall the basic definitions of CSP while fixing the notations used in this paper. Section 3 describes maxRPC3 framework and introduces two implementations of maxRPC3 and their light versions. Then, most importantly, we will expound the new algorithm of maxRPC. In section 4, the algorithm employs a more efficient search strategy. By the end of this paper we will analyze our experimental results and come to conclusion.

## 2 Preliminary Knowledge

Generally speaking, researchers use a triple $(X, D, C)$ to describe a CSP, in which $X$, $D$ and $C$ represent a finite set of variables, the set of domains and a finite set of constraints respectively. Each domain in $D$ $(x_i) \in D$ denotes the current domain of $x_i$, i.e. it denotes the possible values for $x_i$. For each constraint $c \in C$, there is the corresponding subset of variables of $X$, which called vars $(c)$ and represent the scope of $c$. Furthermore, its associated relation is called rel $(c)$ and specifies the values for the variables in vars $(c)$. An assignment of all variables such that each constraint in this problem could find a support could be seen as a solution of a CSP.
2.1 Binary CSP

If each constraint of a CSP involves at most two variables we call it a binary CSP. We assume that all the constraints are binary and binary constraint checks are performed in constant time in our experiment. In a binary CSP, a value \( a_i \in D(x_i) \) is arc consistent (AC) iff for every constraint \( c_{ij} \) there exists a value \( a_j \in D(x_j) \) s.t. the pair of values \((a_i, a_j)\) satisfies \( c_{ij} \).

2.2 Related Work

MaxRPC has attracted a lot of research effort. Two categories: coarse-grained algorithms (i.e. constraint-based) and fine-grained algorithms (i.e. value-based) constitute the existing propagators. The following will give a brief description of these maxRPC algorithms which ensure that each value in the domain of each variable is supported by some value in the domain of a variable. The value is constrained.

- maxRPC1. maxRPC1 [1] maintains a list, which is used to record values that have been removed from the variable universe but not yet propagated. It is a fine-grained algorithm based on AC6 and has optimal \( O(end^3) \) time complexity but requires \( O(ed) \) space complexity.
- maxRPC2. maxRPC2 [3] uses the thought like that of AC2001. MaxRPC2 removes the complicated data structure in maxRPC1 and maintain a list, which is used to record the variables whose fields altered. It is a coarse-grained algorithm having \( O(end^3) \) time and \( O(ed) \) space complexity.
- maxRPC3. maxRPC3 [5] utilizes 2 data structures to record the latest discovered PC-supports and AC-supports for each value on each constraint respectively. MaxRPC3 avoids some redundant constraint checks in maxRPC2.
- maxRPC^rm. maxRPC^rm [4] is based on AC3^rm and use a light data structure, which is like that of MaxRPC2. It reduces the maintenance cost of the data structure and can be well used in the search process. It has \( O(en^d^4) \) time and \( O(end) \) space complexity.
- maxRPC3^rm. maxRPC3^rm [5] use the same data structures as maxRPC3 to record PC and AC supports. Algorithm maxRPC3 updates the 2 data structures incrementally like maxRPC2 and AC2001 [6] respectively do. However, in maxRPC3^rm, these structures are used as residues [7, 8] like maxRPC^rm and AC^rm do.

Hence it is obvious that there are a lot of improvements have been achieved by: using the lighter data structures to reduce the maintenance cost, using new structures to store more information which is more useful, storing the latest support found to improve the performance of algorithms, etc [9].
3 maxRPC3-Framework and Two Implementations

3.1 maxRPC3: A coarse-grained Framework

MaxRPC3 utilizes data structures LastPC and LastAC which have the following functionalities: For each constraint \( c_{ij} \) and each value \( a_i \in D(x_i) \), LastPC\(_{x_i,a_i,x_j}\) and LastAC\(_{x_i,a_i,x_j}\) give (point to) the lastest discovered PC and AC supports of \( a_i \) in \( D(x_j) \) respectively. Initially, all LastPC and LastAC pointers are set to a special value NIL, considered to precede all values in any domain. Algorithm maxRPC3 updates the LastPC and LastAC structures incrementally like maxRPC2 and AC2001/3.1 respectively do. Being coarse-grained, maxRPC3 maintains a propagation list \( L \). Variables that have their domain filtered are inserted in the list \( L \). If the algorithm is used for preprocessing, each value \( a_i \) of each variable \( x_i \) should be checked whether \( a_i \) is maxRPC or not during an initialization phase. If it is not then it is deleted from \( D(x_i) \) and \( x_i \) is added to \( L \). On the contrary, if the algorithm is used in the search, \( L \) will be initialized with the currently assigned variable [5].

3.2 Light maxRPC: The Light version of maxRPC

An approximation of maxRPC is light maxRPC (lmaxRPC). It only propagates the loss of AC-supports but not the loss of PC-witnesses [10]. The obtained algorithm enforces a consistency property that is not stronger than AC is ensured. The description of lmaxRPC is tied to a specific maxRPC algorithm for being a defined local consistency. Experiments with random problems applied algorithms lmaxRPC\(_m\) and maxRPC\(_m\) showed that the pruning power of lmaxRPC is only slightly weaker than that of maxRPC [11]. Meanwhile, when used during search, it can offer significant gains in run times. A series of experiments on various problem classes lead us to these results.

3.3 Performance Comparison Of Two Algorithms

To begin with, we named the processes of searching and checking PC-witnesses as schPCwit and cckPCwit respectively. LmaxRPC3 and LastPC show the main difference is that LastPC is not updated incrementally, for each sought PC-support for a value \( a_i \in D(x_i) \) in \( x_i \), \( D(x_i) \) will be searched from scratch in the worst case. This incurs an extra \( O(d) \) cost to the process of searching PC-supports. Hence, the node complexity of lmaxRPC\(_m\) is \( O(\text{end}^4) \). At the same time, the total cost of searching PC-witnesses in every node cannot be amortized which lead to that the cost of schPCwit within cckPCwit is \( O(\text{nd}^2) \). Hence, the node complexity of maxRPC\(_m\) is \( O(\text{en}^2d^4) \). Because the calls to schPCwit are amortized, the branch complexities are the
same. The space required for data structures LastPC and LastAC determine the space complexities of the algorithms. This is the node space complexity of (l) maxRPC3 and (l) maxRPC3rm because they both require O(ed) space. With extra space required for the incremental update and restoration of the data structures (l) maxRPC3 has O(ed) branch space complexity. (l) MaxRPC3rm avoids part of this, so its branch space complexity is O(ed).

4 MaxRPC-Alter: With a New Search Strategy

In section 3, we have described that among all the performances of maxRPC3, the constraint check needs to be implemented to estimate if \(((x_i, a), (x_k, c)) \in \text{rel}(c_{ik})\) and \(((x_j, b), (x_k, c)) \in \text{rel}(c_{jk})\), while testing if a value \((x_k, c)\) is the PC-witness of the value pair \(((x_i, a), (x_j, b))\). We have to test the value of \(x_k\) separately till the finding of a PC-witness. This method is not quite business-like. For now, for the sake of improving the performance of existing algorithm, we put forward a novel algorithm whose name is (l) maxRPC3-Alter through a more business-like search strategy which is levered. In view of our computers are provided with the 64-bit processor. On the basis of the binary representations in [12], we employ bitwise operation to seek for the PC-witness rather than employ the constraint check, which can test 64 values synchronously every time. Moreover, our novel algorithm still keeps the merits of existing algorithms. Two data structures bs and bd are developed by us to present the constraint and the domain. Bs \([c_{ij}, x_i, a]\) stands for the binary representation of the favors of \((x_i, a)\) which are in the \(c_{ij}\). Every bit, which is in bs \([c_{ij}, x_i, a]\), marks if the corresponding value is a backing of \((x_i, a)\) that exists in the domain of \(x_i\). What’s more, every bit, which is in bd \([X]\), records if the corresponding value exists in the domain of \(X\). Provided that we set one bit to 1 in bd \([X]\), then the corresponding value will appear in the domain. The major ways of maxRPC3-Alter are searchPCsup (see Function 1) and checkPCwit (see Function 3). searchPCsup identifies whether \((x_i, a)\) has a PC-support in \(D(x_i)\). MaxRPC3-Alter begins to seek from the succeeding value of LastPC\(_{x_i}\), because \(a\) and \(x_j\) are the minimum PC-support of \((x_i, a)\). When the maxRPC3rm-Alter seeks from the beginning value in \(D(x_i)\), this is due to LastPC serves as a residual, as well as it does not keep track of the minimum PC-support in maxRPC3rm-Alter for ever. Function IsTrue is used for seeking a value \((x_j, b)\) so that we can get the result of \(((x_i, a), (x_j, b)) \in \text{rel}(c_{ij})\) and return true provided that the corresponding bit of b is 1 which is in bs \([c_{ij}, x_i, a]\). Provided that we find such a value, with regard to every third variable \(x_k\), searchPCwit (see Function 2) is used for establishing if at the fewest one PC-witness of \(((x_i, a), (x_j, b))\) consists in \(D(x_k)\). Provided that the result is unlike to this, Witness will be set to false; the value will be examined, which is after \(b\) in \(D(x_i)\). While we find a PC-support \(b\), at that time, maxRPC3-Alter will set LastPC\(_{a_{l,a,x_j}}\) to \(b\). Just like the maxRPC3rm, it develops the
residue’s multi-directionality, it also sets LastPC\textsubscript{xi,a,xj} to b and LastPC\textsubscript{xj,b,xj} to a. Nevertheless, it does not set LastAC\textsubscript{xi,a,xj} to b like maxRPC\textsubscript{3rm} does, just like in line 13, LastAC\textsubscript{xi,a,xj} is set to the location of b in bs. searchPCwit examines if at the fewest one PC-witness of ((x\textsubscript{i}, a), (x\textsubscript{j}, b)) consists in D (x\textsubscript{k}). During such course, this way which is presented in Section 3 is used. In maxRPC\textsubscript{3rm}-Alter, we will check the surplus position first. While we detect one PC-witness, the location of it which is in bs will be registered. checkPCwit is used to examine for the reduction of PC-witness. With regard to every variate x\textsubscript{k} that is compulsory by not only x\textsubscript{i} but also x\textsubscript{j}, provided that LastPC\textsubscript{xi,a,xk} are effective, it will check if there still exists a PC-witness of the value pair ((x\textsubscript{i}, a), (x\textsubscript{k}, LastPC\textsubscript{xi,a,xk})) which is in D (x\textsubscript{i}). Provided that such a PC-witness exists, LastPC\textsubscript{xi,a,xk} will still be the PC-support of (x\textsubscript{i}, a). Provided that LastPC\textsubscript{xi,a,xk} is noneffective or it is not the PC-support of (x\textsubscript{i}, a) any more, like in line 6 to 8, we have to make sure that if other values in D (x\textsubscript{k}) could be sought out for the PC-support of (x\textsubscript{i}, a).

By means of removing a number of superfluous constrain inspections, maxRPC\textsubscript{3}-Alter possesses the best O (end\textsuperscript{3}) time complexity, and it shows good performance while it is used independently, yet it is costly to use in time of searching. The time complexity of lmaxRPC\textsubscript{3m}-Alter and maxRPC\textsubscript{3m}-Alter is alike to the maxRPC\textsubscript{3m} and lmaxRPC\textsubscript{3m} separately, which are O (end\textsuperscript{4}) and O (en\textsuperscript{d}d\textsuperscript{4}) separately while used independently. Moreover, while lmaxRPC\textsubscript{3m}-Alter is used in time of searching, it possesses a time complexity of O (end\textsuperscript{4}) as well. The supererogatory space leads to the overheads of O (ed\textsuperscript{2}) which is needed by the binary representations.

5 Experimental Results

For the sake of comparing the performance of diverse methods precisely, we have performed all of the proposed algorithms by C++. Moreover, we have also implemented many experiments about series of structured CSP questions which are used as benchmarks and can be seen on the website: http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html, as well as involve not only satisfiable but also unsatisfiable examples. Eliminating examples which were very difficult to all the algorithms, our assessment was accomplished amount to 200 examples from varieties of question classes. Those examples come from diverse backgrounds and tell the structured examples from the arbitrary examples (the entropy of an example will be lower if it is more structured). In our experiments, the performances have been sounded in the light of the CPU time in seconds (t), as well as the amount of constraint checks (cc). Moreover, the consequences of table 1, table 2 and table 3 were implemented on an Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz 3.10GHz which
using the Windows 7. Please be advised average consequences for all the examples are ranged into specific question classes.

The performance of the algorithm, which is used independently, is compared in Table 1. We provide average consequences for all the examples which are sorted into particular question classes. Moreover, we contain consequences from the coarse-grained maxRPC algorithm, maxRPC3 and maxRPC3\textsuperscript{m}, and from the whole coarse-grained algorithm’s light versions, and our competitive algorithm (maxRPC\textsuperscript{Alter} and maxRPC3\textsuperscript{m\textsuperscript{m}}-Alter). The result shows that, in the light of the run time, our algorithm has alike performance and is better than other existing algorithm through an element of three on average. This is because the remove of a number of constraint checks when the cc numbers display. We have compared the performance of search algorithm which applies \textit{lmaxRPC3\textsuperscript{m\textsuperscript{m}}} Alter and \textit{lmaxRPC3\textsuperscript{m}} as well. This two search algorithms employ the dom/wdeg [11] heuristic.

We can know that, from the results from Tables 2 and 3, \textit{lmaxRPC3\textsuperscript{m\textsuperscript{m}}} Alter always transcend \textit{lmaxRPC3\textsuperscript{m}}, and in all the 200 examples which we have tried, the situation is the same. When we look at the columns of \textit{lmaxRPC3\textsuperscript{m\textsuperscript{m}}} Alter and \textit{lmaxRPC3\textsuperscript{m}} we can clearly see that our approaches are able to decrease the number of constraint check by means of in the ballpark one order of magnitude (e.g. in quasigroup questions qcp and qwh). This is mostly because of the remove of parity check in the function \textit{searchPCwit}. Cpu time is not reduced by the same amount, yet it can obtain a more than three times’ acceleration (e.g. scen2-f25 and scen11-f8). It is important that, the accelerations acquired are able to make search algorithms that effectively apply \textit{lmaxRPC3-\textit{Alter}} rival with MAC on a lot of examples. For example, in scen11-f10, we can obtain the same run time like MAC when \textit{lmaxRPC3\textsuperscript{m}} is 3 times slower; we start as two times slower than it, and then we become two times faster than it when in the scen11-f7. Moreover, there are some examples in which MAC is transcended (e.g. the graph RLFAPs and most quasigroup questions). Sure, in spite of the ameliorations, there still exist examples where MAC is still fairly fast.

**Function 1. searchPCsup(x\textsubscript{i}, a, x\textsubscript{j}):boolean**

1: \textbf{if}¬\textit{RM} and \textit{LastPC}_{\textit{i}, a, \textit{x}} ≠ \textit{NIL} \textbf{then}
2: \hspace{1em}v = \textit{LastPC}_{\textit{i}, a, \textit{x}} + 1;
3: \textbf{else}
4: \hspace{1em}v = \textit{first value in D(x\textsubscript{i})};
5: \hspace{1em}\textbf{for each}\ b ∈ D(x\textsubscript{i}), b ≥ v \textbf{do}
6: \hspace{2em}\textit{PCwitness}=true;
7: \hspace{1em}\textbf{if} \textit{IsTrue}(bs[c\textsubscript{ij}, x\textsubscript{i}, a], b) \textbf{then}
8: \hspace{2em}\textbf{for each}\ x\textsubscript{k} ∈ X\ s.t. c\textsubscript{ik} ∈ C and c\textsubscript{jk} ∈ C \textbf{do}
9: \hspace{3em}\textbf{if} ¬\textit{checkPCwit}(x\textsubscript{i}, a, x\textsubscript{j}, b, x\textsubscript{k}) \textbf{then}
10: \hspace{4em}\textit{PCwitness}=false; \textbf{break};
11: \hspace{2em}\textbf{if} \textit{PCwitness} ≠ false \textbf{then}
12: \hspace{3em}\textit{LastPC}_{\textit{i}, a, \textit{x}} = b;
Function 1. `searchPCsup(x_i, a, x_j)`: boolean

13: if RM then
14: \[\text{LastPC}_{x_i, a, x_j} = a; \text{LastAC}_{x_i, a, x_j} = b \div 64;\]
15: return true;
16: return false;

Function 2. `searchPCwit(x_i, a, x_j)`: boolean

1: for each \(x_k \in X\) s.t. \(c_{ik} \in C\) and \(c_{jk} \in C\) do

2: \(\text{maxRPCsupport} = \text{false};\)

3: if \(\text{LastPC}_{x_i, a, x_k} \in D(x_k)\) then

4: if \(\text{checkPCwit}(x_i, a, x_k, \text{LastPC}_{x_i, a, x_k}, x_j)\) then

5: \(\text{maxRPCsupport} = \text{true};\)

6: if \(\text{maxRPCsupport}\) and exists \(c > \text{LastPC}_{x_i, a, x_k} \in D(x_k)\) then

7: if \(\text{searchPCsup}(x_i, a, x_k)\) then

8: \(\text{maxRPCsupport} = \text{true};\)

9: if \(\sim \text{maxRPCsupport}\) then return true;

10: return true;

6 Conclusion

In this paper, we presented maxRPC3-Alter and maxRPC3rm-Alter, two new algorithms for maxRPC, and their light versions that approximate maxRPC. These algorithms build on and improve existing maxRPC algorithms, achieving the elimination of some redundant constraint checks. We also investigated heuristics that can be used to order certain operations within maxRPC algorithms. Experimental results from various problem classes demonstrate that our best method, lmaxRPC3rm-Alter, constantly outperforms existing algorithms, often by large margins. Significantly, the speed-ups obtained allow lmaxRPC3rm to compete with and outperform MAC on many problems. In the future, we will do further research to improve the performance of our algorithms by adapting techniques for using residues from [8].
Function 3. checkPCwit(x_i, a, x_j, b, x_k): boolean

1: if RM then
2:   if LastACx_i, a, x_k ≠ NIL then
3:     i = LastACx_i, a, x_k;
4:       if (bs[c_{ik}, x_i, a][i] AND bs[c_{jk}, x_j, b][i] AND bd[x_k][i]) ≠ 0
5:         then return true;
6:     if LastACx_j, b, x_k ≠ NIL then
7:       j = LastACx_j, b, x_k;
8:       if (bs[c_{ik}, x_i, a][j] AND bs[c_{jk}, x_j, b][j]
9:         AND bd[x_k][j]) ≠ 0
10:       then return true;
11:       for each i ∈ {0, ..., bs[c_{ik}, x_i, a].length-1} do
12:       if (bs[c_{ik}, x_i, a][i] AND bs[c_{jk}, x_j, b][i]
13:         AND bd[x_k][i]) ≠ 0 then
14:         if RM then
15:           LastACx_i, a, x_k = i; LastACx_j, b, x_k = i;
16:         return true;
17:   return false;

Tab. 1: Average stand-alone performance in all 200 instances grouped by problem class. Cpu times (t) in secs and constraint checks (cc) are given.
<table>
<thead>
<tr>
<th>Problem class</th>
<th>maxRPC3</th>
<th>lmaxRPC3</th>
<th>lmaxRPC3-Alter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.054</td>
<td>0.059</td>
<td>0.048</td>
</tr>
<tr>
<td>(modelB,forced)</td>
<td>0.09M</td>
<td>0.10M</td>
<td>0.09M</td>
</tr>
<tr>
<td>Geometric</td>
<td>0.029</td>
<td>0.025</td>
<td>0.02</td>
</tr>
<tr>
<td>Quasigroup</td>
<td>0.06</td>
<td>0.064</td>
<td>0.06</td>
</tr>
<tr>
<td>(qcp,qwh,bqwh)</td>
<td>0.19M</td>
<td>0.19M</td>
<td>0.20M</td>
</tr>
<tr>
<td>QueensKnights,</td>
<td>17.13</td>
<td>16.304</td>
<td>15.736</td>
</tr>
<tr>
<td>Queens, QueenAttack</td>
<td>69M</td>
<td>73M</td>
<td>65M</td>
</tr>
<tr>
<td>Drive, blackHole</td>
<td>0.295</td>
<td>0.338</td>
<td>0.303</td>
</tr>
<tr>
<td>Haystacks, job-shop</td>
<td>0.36M</td>
<td>0.35M</td>
<td>0.34M</td>
</tr>
</tbody>
</table>

**Tab. 2:** Cpu times (t) in secs and constraint checks (cc) from RLFAP instances. The best cpu time among the lmaxRPC methods is highlighted

<table>
<thead>
<tr>
<th>instance</th>
<th>AC&lt;sup&gt;rm&lt;/sup&gt;</th>
<th>lmaxRPC3&lt;sup&gt;rm&lt;/sup&gt;</th>
<th>lmaxRPC3&lt;sup&gt;rm&lt;/sup&gt;-Alter</th>
</tr>
</thead>
<tbody>
<tr>
<td>scen11-f10</td>
<td>t 11.0</td>
<td>12.5</td>
<td>4.1</td>
</tr>
<tr>
<td>cc 11M</td>
<td>51M</td>
<td>20.62M</td>
<td></td>
</tr>
<tr>
<td>scen2-f25</td>
<td>t 27.1</td>
<td>43.0</td>
<td>9.0</td>
</tr>
<tr>
<td>cc 44M</td>
<td>151M</td>
<td>49.15M</td>
<td></td>
</tr>
<tr>
<td>scen11-f8</td>
<td>t 521.1</td>
<td>878.8</td>
<td>172.7</td>
</tr>
<tr>
<td>cc 638M</td>
<td>3,172M</td>
<td>0.35M</td>
<td></td>
</tr>
<tr>
<td>graph8-f10</td>
<td>t 16.4</td>
<td>9.1</td>
<td>5.1</td>
</tr>
<tr>
<td>cc 14M</td>
<td>31M</td>
<td>12.85M</td>
<td></td>
</tr>
<tr>
<td>graph14-f28</td>
<td>t 31.4</td>
<td>3.1</td>
<td>3.0</td>
</tr>
<tr>
<td>cc 13M</td>
<td>8M</td>
<td>4.51M</td>
<td></td>
</tr>
<tr>
<td>graph9-f9</td>
<td>t 273.5</td>
<td>101.5</td>
<td>30.2</td>
</tr>
<tr>
<td>cc 158M</td>
<td>292M</td>
<td>1.00M</td>
<td></td>
</tr>
</tbody>
</table>

**Tab. 3:** Cpu times (t) in secs and constraint checks (cc) from various instances
An Optimized MaxRPC Algorithm with a new Search Strategy

<table>
<thead>
<tr>
<th>instance</th>
<th>AC^m</th>
<th>lmaxRPC^m</th>
<th>lmaxRPC^m-Alter</th>
</tr>
</thead>
<tbody>
<tr>
<td>qwh20-166-1</td>
<td>cc</td>
<td>124M</td>
<td>250M</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>52.4</td>
<td>38.9</td>
</tr>
<tr>
<td></td>
<td>cc</td>
<td>19M</td>
<td>23M</td>
</tr>
<tr>
<td>qwh20-166-6</td>
<td>t</td>
<td>1673.5</td>
<td>867.1</td>
</tr>
<tr>
<td></td>
<td>cc</td>
<td>158M</td>
<td>566M</td>
</tr>
<tr>
<td>qwh20-166-9</td>
<td>t</td>
<td>41.5</td>
<td>25.0</td>
</tr>
<tr>
<td></td>
<td>cc</td>
<td>15M</td>
<td>15M</td>
</tr>
<tr>
<td>blackHole</td>
<td>t</td>
<td>1.5</td>
<td>3.8</td>
</tr>
<tr>
<td>4-4-e-8</td>
<td>cc</td>
<td>4M</td>
<td>12M</td>
</tr>
<tr>
<td>queens-100</td>
<td>t</td>
<td>15.3</td>
<td>106.7</td>
</tr>
<tr>
<td></td>
<td>cc</td>
<td>6M</td>
<td>377M</td>
</tr>
</tbody>
</table>

**Acknowledgement:** The authors would like to express sincere thanks to our instructor, Professor Li because of his careful guidance and valuable suggestions in the process of topic selection and paper writing. Besides, our research is supported by the National Undergraduate Training Programs for Innovation and Entrepreneurship as a national project. We are grateful for this opportunity.

**References**


