Constraint Satisfaction Problem Using Modified Branch and Bound Algorithm

AZLINAH MOHAMED, MARINA YUSOFF, ITAZA AFIANI MOHTAR,
SOFIANITA MUTALIB, SHUZLINA ABDUL RAHMAN
Faculty of Information Technology & Quantitative Sciences
Universiti Teknologi MARA
40450 Shah Alam, Selangor
MALAYSIA
azlinah@tmsk.uitm.edu.my, marinay@tmsk.uitm.edu.my,
shuzlina@tmsk.uitm.edu.my,
sofi@tmsk.uitm.edu.my

Abstract: - A constraint satisfaction problem (CSP) involves assigning possible values to a set of variables without defying any constraints. There are various techniques available to solve or give partial solution to CSP. This paper presents a modification of branch and bound algorithm, which is used to solve a constraint satisfaction problem in map colouring problem. There are two constraints involved which are only three colours are allowed to be used and adjacent regions in the map must not be of the same colour. The modified branch and bound algorithm uses back jumping when it encounters a dead-end in the search. Static variable ordering was also applied to aid the searching process. The modified branch and bound algorithm shows a better result in terms of the number of nodes instantiated and the reduced number of backtracking at dead ends. The result illustrated that the modified branch and bound algorithm with the use of variable ordering technique is better if compared to backjumping. Thus, it is concluded that the modified branch and bound algorithm would improve constraint satisfaction problem.

Key-Words: - Backjumping, Branch and Bound Algorithm, Constraint Satisfaction Problem, and Static Variable Ordering

1 Introduction

Constraints are used in everyday life to guide reasoning. Naturally, humans do not solve one constraint but a collection of constraints that are rarely independent. Examples of constraints represented in daily life are restrictions, requirements, regulations, preferences and machine capacity, to name a few. Constraint Satisfaction Problem (CSP) is a technique where one has to find a value for a finite set of variables satisfying a finite set of constraints [1]. In other words, the complete solution to the problem would be to assign possible values to the set of variables that do not defy any constraints [2].

There exist quite a number of techniques used in solving CSP [3, 4]. Each has its own advantages and disadvantages. They are binarisation of constraints, systematic search algorithms, consistency techniques, constraint propagation, variable & value ordering, reducing search, and heuristics & stochastic algorithms [5, 6, 7, 8]. From these techniques, branch and bound (B&B) algorithm was chosen for this study due to its capability in searching complete space of solutions for best solution. Possible modifications to the B&B algorithm was done and applied to solve a CSP problem and the results obtained were critically analysed.

2 Branch and Bound Algorithm

B&B algorithm is a general search method and may be among the most widely used algorithm for finding optimal solutions [9]. It is a depth-first search where internal nodes represent incomplete assignments and leaf nodes represent complete ones. B&B uses a heuristic function, *h* that approximates the objective function, *f*. A heuristic for a minimization problem represents an underestimate, $h(x) <= f(x)$ whereas for a maximization problem it would be represented by an overestimate, $f(x) <= h(x)$.

B&B algorithm would map every complete labeling of variables (solution) to a numerical value. At the start of the search, the value for the bound is set to infinity. As the search proceeds, the bound

would be set to the value of the best solution found so far. B&B algorithm would perform a depth-first traversal through the search tree. It uses chronological backtracking [10] when it encounters a dead end but at the same time, it would also compute the value of the heuristic function for the labeling. If the value exceeds the bound then the subtree would be pruned. The search proceeds until all nodes have been solved or pruned, or until some specified threshold is met between the best solutions found and the bound. This algorithm would be efficient if it is represented by a good heuristic function and a good bound [11]. The algorithm for B&B is as follows.

Step 1: Initialize the upper bound to 0.

Step 2: Check to see if the leaf node has been reached. If true then the best solution is found currently. If it is not a leaf node, then go to step 3.

Step 3: Choose the next variable from the list. Assign a value and check with upper bound. If it is above the upper bound, then prune the branch and perform backtracking. Repeat step 3 until search reaches the leaf node.

Figure 1 Algorithm for B&B

3 Modified B&B Algorithm

The B&B algorithm is a depth-first search using chronological backtracking. When using chronological backtracking, the algorithm would generate sub-trees that are identical to previously explored sub-trees. This problem would contribute to the inefficiency of the search. Three modifications were introduced to overcome this problem. The first modification is by combining B&B with static variable ordering. The second modification is by combining B&B with backjumping and the third is by combining B&B with static variable ordering and back jumping. Before detailing the combinations, static variable ordering and back jumping would be explained.

3.1 Static Variable Ordering

According to Bartak [11], the order in which variables are chosen for instantiations can have an impact on the complexity of backtrack search. Static variable ordering specifies that the variables be ordered before the search begins. The degree of the heuristic was determined as the heuristic value for selecting the variable ordering.

This heuristic chooses the variable that has the largest number of constraints with the past variable as the variable to be instantiated next. The coloured adjacent state would be the next variable to be instantiated. The search would then proceed among its neighbours. The neighbour with the most constraint would be chosen next. This would go on until all the variables are assigned. This is beneficial in the long term because it can reduce the average depth of branches in a search tree. The algorithm involved in the static variable ordering [12] is as shown in figure 2.

Step 1: The variable with the most constraints is specified and put in front of the list, List l. All the other variables would also be collected and put into List l.

Step 2: The variables would then be transferred into List 2 one by one. During the process, the neighbors associated with each variable transferred into List 2 would be included in List 1. A check is made to ensure that there is no duplicate data in List 2. This is done when considering the next variable to transfer to List 2. If it has been considered before, the variable would be discarded.

Step 3: Variables in List 2 can now be used to generate the search tree.

Figure 2 Algorithm static variable ordering

3.2 Backjumping

Backjumping is an intelligent version of the chronological backtracking. Instead of backtracking to the parent node (as displayed by chronological backtracking), backjumping jumps to the highest node that conflicted with the current variable. This can reduce the amount of thrashing[1].

Other than that, the overhead cost is small in maintaining the consistency checks done to

[1] generating sub-trees that are identical to previously explored sub-trees by instantiating variables that play no role in the current inconsistency

determine the backtracking point. The algorithm for backjumping [13] is as demonstrated in figure 3.

Step 1: If X is the last variable then all variables have value assignments. Exit with this solution. Else proceed to Step 2.

Step 2: Set *Current* equal to the next index in variable ordering. Set array *J* to O. This array would store the latest variable in the ordering that was tested for consistency with variable X.

Step 3: Select a value for X which is consistent with all previous variables. If there is no suitable value then proceed to step 4. For $1 < i <$ Current, if $i >$ J cur then set J_{cur} to *i*. If value for the current variable is equal to the value of past variable, then choose the next value. If all values were chosen then proceed to step 4.

Step 4: (Backjump). Select variable $X_{J\ cur}$ as the next variable to assign a value. Go to step 3.

Figure 3 Algorithm for backjumping

3.3 B&B with Static Variable Ordering

This algorithm starts with the ordering procedure. The static variable ordering algorithm in Section 3.1 is used. The new ordered list of variables would then be the initial variables for the search. The search would use the B&B algorithm as stated in Section 2 above.

3.4 B&B with Backjumping

Whenever the B&B algorithm discovers a dead-end, it would backtrack. This backtracking can be changed into back jumping with a few alterations. Figure 4 shows the modified algorithm of B&B with Backjumping.

This algorithm has an array that stores the latest variable in the ordering list that conflict with the current value. It would perform consistency checks each time a variable is instantiated with a value. If there is an inconsistency, then it would backjump to the variable in the list according to the index of the inconsistent variable. Backjumping would only occur if there is a dead end. If the inconsistency can be eliminated with a change of value, then normal backtracking is sufficient enough.

Step I: Initialize the upper bound to 0.

Step 2: Check to see if the leaf node has been reached. If true then the best solution is found currently. If it is not a leaf node, then go to step 3.

Step 3: Set array *J* to 0. This array would store the latest variable in the ordering that was tested for consistency with variable X.

Step 4: Choose the next variable from the list. Assign a value and check with upper bound. If it is equal to the upper bound, repeat step 4. If it is above the upper bound, then go to step 5.

Step 5: If value for the current variable is equal to the value of past variable, then choose the next value. If all values were chosen and variable remains inconsistent then proceed to step 6.

Step 6: (Backjump). Select variable $X_J cur$ as the next variable to assign a value. Go to step 4.

Figure 4 Modified algorithm of B&B and backjumping

3.5 B&B with Static Variable Ordering with Backjumping

Before the search begins, the variables are ordered using the static variable ordering algorithm as described in Section 3.1 above. The ordered list is then used as the initial variables for the B&B tree. Whenever the algorithm encounters a dead end, it would use back jumping. The algorithm for this procedure is explained in Section 3.4 above.

4 Result and Analysis

The modified algorithms discussed in Section 3 were compared to the B&B algorithm without any modification. The comparison is based on whether the algorithms can solve the same map colouring problem, the time taken for each algorithm to solve the problem, the number of backtracking at dead-ends that occurred and the number of nodes instantiated. Prolog built-in predicate ms/2 is used to time the search. The predicate calls the goal and returns the duration of its execution. The modified algorithms have been tested to instantiate states in

West Coast of Malaysia. The results are as discussed in the next section.

4.1 B&B Algorithm

The B&B algorithm was able to solve the problem in approximately 7 milliseconds (ms) with three backtracking at dead-end. The result is shown in Table 1 while the traversal of the search space together with backtracking is shown in Figure 5. B&B algorithm would perform chronological backtracking when dead-end is encountered. However, this had a very small impact on the performance. Backtracking had to be done a few times before the algorithm reached the same nodes of the conflict which is situated four levels above the inconsistent node.

Table 1 Result of B&B algorithm

Order of variables	Number of Backtracking at dead ends	Number of nodes instantiated	Solution
perlis, n_sembilan, johor, selangor, pulau_pinang, w_persekutuan, kedah, perak, melaka, terengganu, kelantan	3	21 (perlis, n_sembilan, johor, selangor, pulau_pinang, w_persekutuan, pahang, kedah, kedah, w_persekutuan, pahang, kedah, kedah, pulau_ pinang, w_persekutuan, pahang, kedah, perak, melaka, terengganu, kelantan)	[kelantan,green] [terengganu,red] [melaka,blue] [perak,red] [kedah,blue] [pahang,blue] [w_persekutuan,red] [pulau_pinang, green] [selangor,green] [johor,green] [n_sembilan,red] [perlis,red]

The number of nodes instantiated was twenty-one (21). This shows that there is a need to include intelligent backtracking in the B&B algorithm, which theoretically can improve the performance of the search and reduce the number of nodes instantiated.

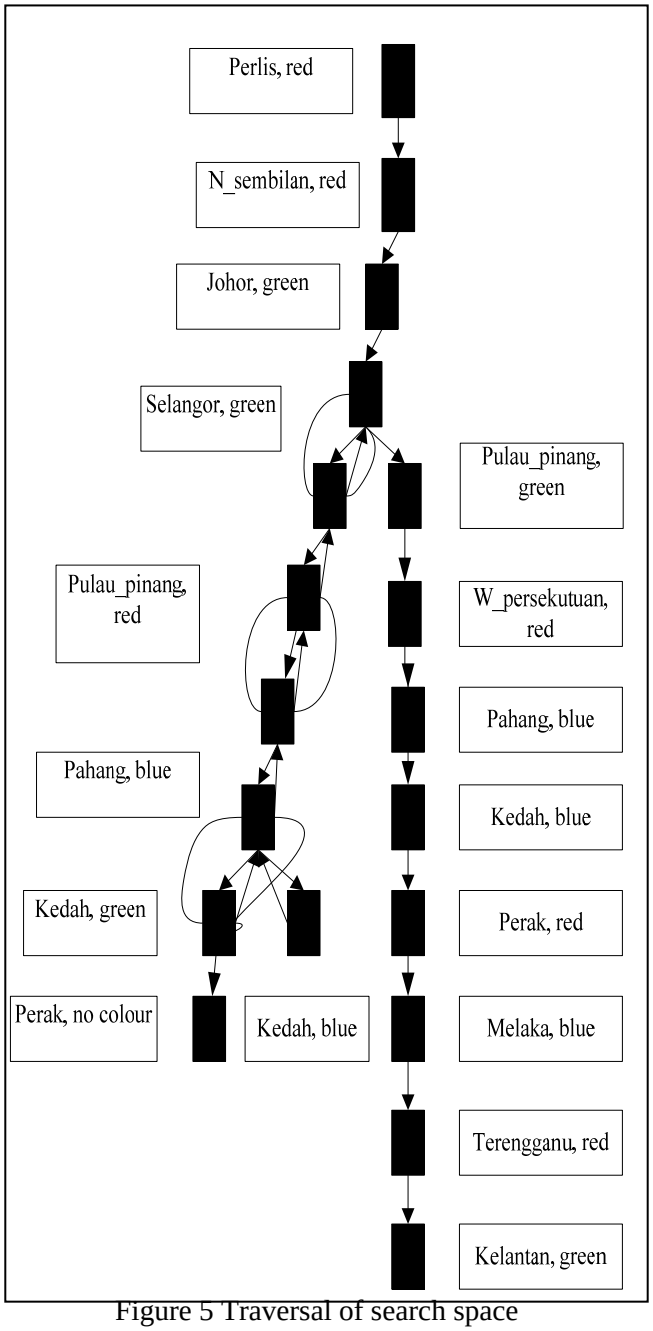

Figure 5 Traversal of search space

4.2 B&B with Variable Ordering

The B&B algorithm was then included with variable ordering. The static ordering ordered the variables according to the variables that had the most number of constraints with the previous variable. The ordering was generated three times to see the effects on the time taken to solve the problem and also the number of nodes instantiated. The result is as illustrated in Table 2.

Table 2 Result of different variable orderings on time, backtracking and number of nodes instantiated

Variable ordering	Time (ms)	Number of backtracking at dead-ends	Number of nodes instantiated
pahang, perak, selangor, n_sembilan, johor, melaka, w_persekutuan, kelantan, terengganu, kedah, pulau_pinang, perlis	9	0	12
pahang, selangor , perak, kelantan, terengganu, kedah, pulau_pinang, perlis, n_sembilan, johor,melaka, w_persekutuan	8	0	12
pahang,johor, n_sembilan, selangor, perak, kelantan, terengganu, kedah, pulau_pinang, perlis, w_persekutuan, melaka	9	0	12

There was not much difference in time recorded as compared to B&B without variable ordering. But there were no backtrackings at dead-ends recorded because the algorithm never reached a dead-end. This in fact reduced the number of nodes instantiated from twenty-one (21) with B&B without any modifications to twelve (12).

This result is rather interesting because the algorithm does not seem to require intelligent backtracking since the algorithm does not encounter dead-ends. As a matter of fact, if intelligent backtracking were to be included, there is a theoretical possibility that the saving occasionally intended for reducing the search space would actually be undone by the overhead of computing and maintaining the extra information.

4.3 B&B with Backjumping

The B&B algorithm was modified with back jumping. Backjumping would only occur when the algorithm reaches a dead-end. It would still perform chronological backtracking when a conflict occurs. This algorithm was able to solve the problem. There was also a very small difference in the time recorded to solve the problem. It took 10 ms to solve the problem. The result of the algorithm is as shown in Table 3.

Table 3 Result of B&B with Backjumping Algorithm

Order of variables	Number of Backtracking at dead ends	Number of Nodes instantiated	Solution
perlis, n_sembilan, johor, selangor, pulau_pinang, w_persekutuan, pahang,kedah, perak,melaka, terengganu, kelantan	1	15(perlis, n_sembilan, johor, selangor, pulau_pinang, w_persekutuan, pahang, kedah, w_persekutuan, pahang, kedah, perak, melaka, terengganu, kelantan)	[kelantan,green] [terengganu,red] [melaka,blue] [perak,red] [kedah,blue] [pahang, blue] [w_persekutuan,red] [pulau_pinang,green] [selangor, green] [johor,green] [n_sembilan,red] [perlis,red]

It came upon a dead-end only once and handled it with back jumping. The number of the nodes instantiated was greatly reduced from twenty one (21) to fifteen (15) as compared to B&B without back jumping because the algorithm did not do chronological backtracking when encountering a dead end.

4.4 B&B with Variable Ordering and Backjumping

The B&B algorithm was modified with the combination of two algorithms, which are variable ordering and backjumping. The algorithm has no problem in solving the map coloring problem. As anticipated, there was no backjumping because the algorithm did not encounter a dead end.

The number of nodes instantiated was smaller when compared to B&B with back jumping because the search did not encounter a dead end. In terms of time taken, the modifications had little effect because there was not much difference recorded. The time it took to solve the problem was approximately nine (9) ms.

The modifications however had little effect in reducing the time to complete the search but it contributed to the reduction of the number of nodes instantiated as shown in Table 4 shows. Solving the problem using B&B with backjumping saw the

number of nodes instantiated greatly reduced from 21 to 15 when compared with B&B without backjumping because the algorithm did not do chronological backtracking when encountering a dead-end. B&B with variable ordering did not require a backjumping procedure because the search never encountered a dead end. The number of nodes instantiated is lesser when compared to B&B with backjumping.

Table 4 Results of each algorithm

Algorithm	Time (ms)	Number of Backtracking at dead-ends	Number of Backjumping at dead-ends	Number of nodes instantiated
B&B	7	3	-	21
B&B + Variable Ordering	9	0	-	12
B&B + Backjumping	10	-	1	15
B&B + Variable Ordering + Backjumping	9	-	0	12

Although there are differences in the variables such as the time taken to solve the problem, the number of nodes instantiated and the number of backtracking at dead ends, the differences are minimal. This is because the problem considered a small number of nodes.

5 Conclusion

The B&B with backjumping algorithm has the ability to jump to the same nodes of inconsistency rather than backtrack chronologically up the branch. After it back jumps, it would prune the search tree and explore other branches that would not create previous inconsistencies. This reduces the search space and adds efficiency to the search. The number of nodes instantiated was also greatly reduced when compared with B&B without backjumping. This modified algorithm only detects inconsistency when it comes across one. It does not predict inconsistencies, which could hypothetically improve search.

B&B with variable ordering did not require a backjumping procedure because the search never encountered a dead end. The number of nodes instantiated is smaller when compared to B&B with back jumping. In terms of the number of nodes instantiated, it can be concluded that B&B with variable ordering is better compared to B&B with backjumping.

These algorithms could be further tested for the whole state of Malaysia including Sabah and Sarawak, cities, and towns. This probably gives more convincing results on the modified algorithms due to the increasing of nodes in traversal search space. Furthermore, considering distances between each state, cities, and towns would be significantly an additional variables to show the ability of the modified algorithm in solving constraints satisfaction problem.

In addition, the algorithm could be further extended to solve other problems beside the map colouring problem to find a generic algorithm. The combination of look ahead and look back techniques can also be applied to the B&B algorithm. With this combination, a dynamic ordering heuristic can be applied as opposed to the static ordering applied in the research. Through dynamic ordering, the order of the variables can be determined during the search. This would make the algorithm more adaptable to situations.

References:-

[1] Freuder, E. C, and Mackworth, A. (ed), Constraint-Based Reasoning, *MIT Press*, 1994. Taken from Tsang, E. A Glimpse of Constraint Satisfaction, *Journal Artificial Intelligence Review*, Vol.13, 1999, pp. 215-227.

[2] Tsang, E, A., Glimpse of Constraint Satisfaction, *Journal Artificial Intelligence Review*, Kluwer Academic Publishers, The Netherlands. Vol. 13, 1999, pp. 215-227.

[3] Baker, A. B., Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Result, *PhD thesis*, University of Oregon, 1995.

[4] Bruynooghe, M., Enhancing a Search Algorithm to Perform Intelligent Backtracking, (online) http://arxiv.org/abs/cs.AIl0311O03, Date accessed 24.12. 2003, 2003.

[5] Dechter, R, and Frost, D., Backtracking Algorithms for Constraint Satisfaction Problems,(online) http://repositOlYZSzpaperszSzsurveyR56.pdf/ dechter99backtracking.pdf, Date accessed 9.9. 2003, 1999.

[6] Frost, D. H., Algorithms and Heuristics for Constraint Satisfaction Problems, Phd Thesis, University of California Irvine, 1997.

[7] Meseguer, P. et al., Current Approaches for Solving Over-constrained Problems, *Journal Constraints*, Kluwer Academic Publishers, The Netherlands, Vol. 8, 2003, pp. 9-39.

[8] Pannee, 1., The integration of evolutionary and adaptive computing technologies with product/system design and realization, *Springer- Verlag*, Plymouth, UK, 1989, Taken from Meseguer, P. et al, Current approaches for solving over-constrained problems. *Journal Constraints*, Kluwer Academic Publishers, The Netherlands, Vol. 8, 2003, pp 9-39.

[9] Lawler, E. W., and Wood, D. E., Branch and Bound Methods: A Survey in Operations Research, Vol.14, 1977, pp. 99-118.

[10]Kondrak, G. A., Theoretical Evaluation of Selected Backtracking Algorithms, (online) http://citeseer.nj.nec.com/43467.html, Accessed date 4.9. 2003, 1994.

[11]Bartak, R., Constraint Programming : In Pursuit of The Holy Grail, (online) http://citeseer.ist.psu.edu/cache/papers/cs/13544/http:zSzzSzwww.insol.co.ilzSzWDS99.pdf/bartak99constraint.pdf , Access date 5.9. 2003, 1999.

[12]Russell, S. I., and Norvig, P., *Artificial Intelligence A Modem Approach*, 2nd Edition, Pearson Education Inc, New Jersey, 2003.

[13]Gaschnig, J. G., Performance Measurement and Analysis of Certain Search Algorithms, PhD Thesis, Carnegie Mellon University, Pittsburgh, 1979.