

Modified Branch and Bound Algorithm

AZLINAH MOHAMED, MARINA YUSOFF, SOFIANITA MUTALIB, SHUZZLINA
 ABDUL RAHMAN
 Faculty of Information Technology & Quantitative Sciences
 Universiti Teknologi MARA
 40450 Shah Alam, Selangor
 MALAYSIA

Abstract: - There are various techniques available to solve or give partial solution to constraint satisfaction problem. 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 in which only three colours are allowed to use 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 nodes instantiated and reduced the 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.

Keywords:- Branch and Bound Algorithm, Backjumping, Constraint Satisfaction Problem, 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]. From these techniques, branch and bound (B&B) algorithm was chosen for this study. 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 [8]. 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, $h(x) \leq f(x)$ whereas for a maximization problem it will be represented by an overestimate, $f(x) \leq h(x)$. 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 will be set to the value of the best solution found so far. B&B algorithm will perform a depth-first traversal through the search tree. It uses chronological backtracking [9] when it encounters a dead-end but at the same time, it will also compute the value of the heuristic function for the labeling. If the value exceeds the bound then the subtree will 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 will be efficient if it is represented by a good heuristic function and a good bound [10].

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 will be explained.

3.1 Static Variable Ordering

According to Bartak [2], the order in which variables are chosen for instantiations can have an impact on the complexity of backtrack search. Static variable ordering specifies that variables are 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

will be the next variable to be instantiated.

The search will then proceed among its neighbours. The neighbour with the most constraint will be chosen next. This will 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 [11].

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 thrashing¹. Other than that, the overhead cost is small in maintaining the consistency checks done to determine the backtracking point [12].

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 will then be the initial variables for the search. The search will 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 will backtrack. This backtracking can be changed into back jumping with a few alterations. Below is the new 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 will perform consistency checks each time a variable is instantiated with a value. If

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

there is an inconsistency, then it will backjump to the variable in the list according to the index of the inconsistent variable. Backjumping will 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.

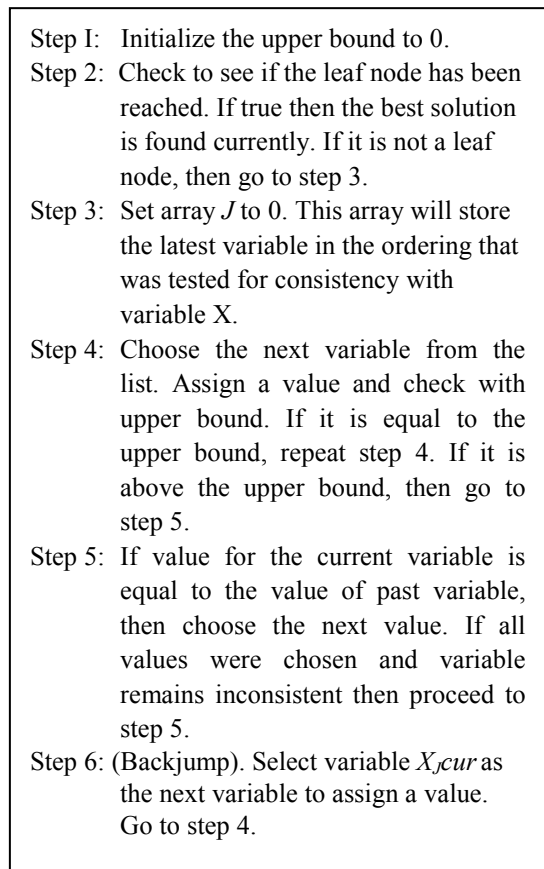


Fig. 1 Modified Algorithm of B&B and Backjumping

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 will 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 Cost of Malaysia. It has covered fourteen (14) states namely; Perak, Negeri Sembilan, Johor, Selangor, Pulau Pinang, Pahang, Kedah, Perak, Wilayah Persekutuan, Melaka, Terenganu, Perlis, Kelantan, and Melaka. The results are as discussed in the next section.

4.1 B&B

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. 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. 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.

Table 1. Result of B&B algorithm

Order of variables	perlis,n - sembilan, johor,selangor, pulau yinang, w yersekutuan, pahang, kedah,perak,melaka, terengganu,kelantan
Number of Backtracking at dead ends	3
Number of nodes instantiated	21 (perils, n sembilan johor, selangor, pulau pinang, w yersekutuan pahang, kedah, kedah wpersekutuan, pahang kedah, kedah, pulau pinang, w_persekutuan, pahang, kedah, perak melaka, terengganu, kelantan }
Solution	[kelantan,green] [terengganu,red] [melaka,blue] [perak,red] [kedah,blue] [pahang,blue] [w yersekutuan,red] [pulau yinang,green] [selangor ,green] [johor,green] [n - sembilan,red] [perlis,red]

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. 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 reduced the number of nodes instantiated from twenty-one (21) with B&B without any modifications to twelve (12).

Table 2. Results 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

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 will only occur when the algorithm reaches a dead-end. It will still perform chronological backtracking when a conflict occurs. The result of the algorithm is as shown in Table 3.

Table 3. Result of B&B with Backjumping Algorithm

Order of variables	perlis, n_sembilanjohor, selangor, pulau_pinang, w_persekutuan, pahang, kedah, perak, melaka, terengganu, keantan
Number of Backtracking at dead ends	1
Number of nodes instantiated	15 (perlis, n_sembilan, johor selangor, pulau pinang w -persekutuan, pahang kedah, w -persekutuan pahang, kedah, perak melaka, terengganu kelantan)
Solution	[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]

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. 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 backjumping 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. Table 4 shows the four criteria considered in comparing the algorithms.

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	0	21
B&B + Variable Ordenng	9	0	0	12
B&B + Backjumping	10	0	1	15
B&B + Variable Ordering + Backjumping	9	0	0	2

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 will prune the search tree and explore other branches that will 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.

Acknowledgment

The authors would like to sincerely thank to our research student, Itaza Afiani Mohtar for her valuable work that support this research.

References:

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

[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 Results", PhD thesis, University of Oregon, 1995.

[4] Bruynooghe, M. (2003). *Enhancing a Search Algorithm to Perform Intelligent Backtracking*. <http://arxiv.org/abs/cs.AI/0311003>, (Dec 24, 2003)

[5] Dechter, R, and Frost, D. (1999) *Backtracking Algorithms for Constraint Satisfaction Problems*, <http://repositOIYZS.zpaperszSzsurveyR56.pdf/dechter99backtracking.pdf>, (Sept 9, 2003)

[6] Frost, D. H. (1997) *Algorithms and Heuristics for Constraint Satisfaction Problems*. Pill Thesis, University of California Irvine. (on line) [http://www.ics.uci.edu/"CspIR69.pdf](http://www.ics.uci.edu/), (Dec 21, 2003)

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

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

[9] Kondrak, G. (1994) *A Theoretical Evaluation of Selected Backtracking Algorithms*. (online) <http://citeseer.nj.nec.com/43467.html>, (Sept 4, 2003)

[10] Bartak, R. (1999) *Constraint Programming : In Pursuit of The Holy Grail*. <http://citeseer.ist.psu.edu/cache/papers/cs/13544/http:zSzzSzwww.insol.co.ilzSzWD599.pdf/bartak99constraint.pdf>, (Sept 5, 2003)

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

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