Abstract. Pseudo-tree search is a well-known algorithm for CSPs (Constraint Satisfaction Problems) and is a search algorithm that exploits the problem structure to make the search more efficient. It is based on the idea of using local upper bounds to prune the search tree.

1 INTRODUCTION

Pseudo-tree search is a well-known algorithm for CSPs. When applied to CSPs, it works by constructing a tree-like structure for the problem and then searching this tree, pruning branches that do not lead to solutions. This approach can be particularly effective for problems with a certain structure, such as those with a small backbone width.

2 PRELIMINARIES

In the following, we will examine the problem of weighted constraint satisfaction problems (WCSPs). WCSPs are a generalization of CSPs where constraints are weighted. The goal is to find an assignment of values to variables that satisfies the constraints and minimizes the total weight.

In this paper, we extend pseudo-tree search to WCSPs by introducing a new algorithm called PT-BB. PT-BB is a branch-and-bound algorithm that performs efficient backtracking. It uses local upper bounds to prune the search tree, which can improve the performance of the algorithm.

3 IMPLEMENTATION

We implemented PT-BB and compared it to other algorithms for solving WCSPs. The results show that PT-BB is competitive with state-of-the-art algorithms, particularly for problems with a small backbone width.

4 EXPERIMENTAL RESULTS

We conducted experiments on a variety of problem instances to evaluate the performance of PT-BB. The results indicate that PT-BB is effective in solving WCSPs, especially for problems with a small backbone width.

5 CONCLUSION

In conclusion, PT-BB is a promising algorithm for solving WCSPs. It is simple to implement and can be effective in many cases. Further research is needed to explore the potential of PT-BB for solving other types of constraint satisfaction problems.
The current problem has a solution if every independent subproblem has a solution. Pseudo-tree search has time complexity $O(|\mathcal{X}|^2)$.

The extension of pseudo-tree search to WCSPs solving requires the constraint graph and the original constraint graph. The simplest idea is to use pseudo-tree search. Starting from the pseudo-tree root, if the current domain of variable is empty, then return the current assignment. Otherwise, if the lower bound is greater than the global upper bound, then return the current assignment. Else, choose a variable and its domain value.

In the current problem, the pseudotree search with the current problem defined by the tuple $(q, \mathcal{D}, \mathcal{F})$ is extended to the constraint graph. Each constraint in the constraint graph is extended to a constraint in the pseudo-tree. The pseudo-tree is a rooted tree with the same set of vertices as the constraint graph. The root is the constraint graph root, and the leaves are the variables in the constraint graph.

The pseudo-tree search works as follows: If the set $\mathcal{X}(t)$ is the current domain of variable, then return the current assignment. Else, if the lower bound is greater than the global upper bound, then return the current assignment. Else, choose a variable and its domain value.

The pseudo-tree search with the current problem defined by the tuple $(q, \mathcal{D}, \mathcal{F})$ is extended to the constraint graph. Each constraint in the constraint graph is extended to a constraint in the pseudo-tree. The pseudo-tree is a rooted tree with the same set of vertices as the constraint graph. The root is the constraint graph root, and the leaves are the variables in the constraint graph.

The pseudo-tree search works as follows: If the set $\mathcal{X}(t)$ is the current domain of variable, then return the current assignment. Else, if the lower bound is greater than the global upper bound, then return the current assignment. Else, choose a variable and its domain value.

In the current problem, the pseudotree search with the current problem defined by the tuple $(q, \mathcal{D}, \mathcal{F})$ is extended to the constraint graph. Each constraint in the constraint graph is extended to a constraint in the pseudo-tree. The pseudo-tree is a rooted tree with the same set of vertices as the constraint graph. The root is the constraint graph root, and the leaves are the variables in the constraint graph.

The pseudo-tree search works as follows: If the set $\mathcal{X}(t)$ is the current domain of variable, then return the current assignment. Else, if the lower bound is greater than the global upper bound, then return the current assignment. Else, choose a variable and its domain value.

The pseudo-tree search with the current problem defined by the tuple $(q, \mathcal{D}, \mathcal{F})$ is extended to the constraint graph. Each constraint in the constraint graph is extended to a constraint in the pseudo-tree. The pseudo-tree is a rooted tree with the same set of vertices as the constraint graph. The root is the constraint graph root, and the leaves are the variables in the constraint graph.

The pseudo-tree search works as follows: If the set $\mathcal{X}(t)$ is the current domain of variable, then return the current assignment. Else, if the lower bound is greater than the global upper bound, then return the current assignment. Else, choose a variable and its domain value.
Figure 3. The algorithm proceeds as follows (RDS) [11, 8]: 

- A subproblem notation is used, which is returned (line 19).
- The lower bound of the cost of the best solution remains unchanged, as there is no solution improvement.
- The function is called if more likelihood that we can improve over the lower bound.
- The subproblem notation is used.
- The algorithm proceeds as follows (RDS) [11, 8]: 

```plaintext
def function(subproblem):
  if more likelihood that we can improve over the lower bound:
    return RDS(subproblem)
  else:
    return None
```

The algorithm is recursively applied to each subproblem, and the process repeats until no further improvements are possible.

**Figure 4.** COMBINING PSEUDO-TREE AND RUSSIAN DOLL SEARCH

Russian Doll Search (RDS) [11, 8] is a BB algorithm that invests in any value of variable \( \text{Var} \), produces subproblems nested from the root to the lower bound. Given a nested subproblem, the cost of the best solution remains unchanged, as there is no solution improvement.

- A lower bound is computed (line 17).
- Once all independent subproblems have been solved, if the lower bound is better than the current best solution, the current subproblem does not need to be solved.
- If the lower bound of the optimal subproblem is greater than or equal to the global lower bound, the algorithm stops.
- The current subproblem is solved recursively.
- The cost of the best solution remains unchanged, as there is no solution improvement.

**Figure 5.** Pseudo-Tree Branch-and-Bound

Pseudo-Tree Branch-and-Bound (PT-BB) searches on nested subproblems, with a cost less than that of the previous subproblem. The cost of the best solution remains unchanged, as there is no solution improvement.

- If the lower bound of the optimal subproblem is greater than or equal to the global lower bound, the algorithm stops.
- The current subproblem is solved recursively.
- The cost of the best solution remains unchanged, as there is no solution improvement.

**Figure 6.** Pseudo-Tree Branch-and-Bound

Pseudo-Tree Branch-and-Bound (PT-BB) searches on nested subproblems, with a cost less than that of the previous subproblem. The cost of the best solution remains unchanged, as there is no solution improvement.

- If the lower bound of the optimal subproblem is greater than or equal to the global lower bound, the algorithm stops.
- The current subproblem is solved recursively.
- The cost of the best solution remains unchanged, as there is no solution improvement.

**Figure 7.** Pseudo-Tree Branch-and-Bound

Pseudo-Tree Branch-and-Bound (PT-BB) searches on nested subproblems, with a cost less than that of the previous subproblem. The cost of the best solution remains unchanged, as there is no solution improvement.

- If the lower bound of the optimal subproblem is greater than or equal to the global lower bound, the algorithm stops.
- The current subproblem is solved recursively.
- The cost of the best solution remains unchanged, as there is no solution improvement.
In this paper, we have explored the relevance of local upper bounds in our implementation, substituting line 10 of Figure 3 by

\textit{algorithm} that performs pseudo-tree search. Its main feature is that it fine-tunes to variable orderings, we decide to construct the pseudo-tree according to the assignment (CELAR) instances \cite{2}.

6 CONCLUSION

Pseudo-tree search is a well-known algorithm for CSP with two nice properties:

- it can be performed on any domain size without expensive preprocessing steps.
- the space complexity is polynomial. In this paper, we will call the algorithm presented in this paper pseudo-tree search.

Figure 6 contains the average number of visited nodes for three problems (a) and (b) to very low connectivity, PT-SRDS is the algorithm of choice. PT-RDS and PT-SRDS both offer the best performance, followed by MRDAC and SRDS in third position. From these results, we conclude that, for random binary problems with medium connectivity classes, both algorithms have extended pseudo-tree search to the soft constraints framework.

We have evaluated the performance of pseudo-tree RDS using the following problem classes (the other results are omitted for space reasons).

- \textit{RDS} and \textit{PT-RDS} are problems with low connectivity, \textit{PT-RDS} exhibits the best performance, followed by MRDAC and SRDS in third position. From these results, we conclude that, for random binary problems with medium connectivity classes, both algorithms have extended pseudo-tree search to the soft constraints framework.

We have explored the relevance of local upper bounds in our implementation, substituting line 10 of Figure 3 by

\textit{algorithm} that performs pseudo-tree search. Its main feature is that it fine-tunes to variable orderings, we decide to construct the pseudo-tree according to the assignment (CELAR) instances \cite{2}.

We have evaluated the performance of pseudo-tree RDS using the following problem classes (the other results are omitted for space reasons).

- \textit{RDS} and \textit{PT-RDS} are problems with low connectivity, \textit{PT-RDS} exhibits the best performance, followed by MRDAC and SRDS in third position. From these results, we conclude that, for random binary problems with medium connectivity classes, both algorithms have extended pseudo-tree search to the soft constraints framework.
We have presented some preliminary empirical results showing that our algorithms are competitive with state-of-the-art solvers. We believe that our results can be further improved, because we have not addressed several important practical aspects. For instance, we have not studied the effect of the ordering in which the independent subproblems are solved. Similarly, we have used naive pseudo-tree arrangements, probably having non-optimal height. The effect of this on the algorithm's efficiency is a topic of our current work.

ACKNOWLEDGEMENTS

This work was supported by the IST Programme of the Commission of the European Union through the ECS-PLAIN project (IST-1999-11969), and by the Spanish CICYT project TAP99-1086-C03. We thank the anonymous reviewers for their constructive criticisms.

REFERENCES