A Hybrid Approach to Distributed Constraint Satisfaction

David Lee, Inés Arana, Hatem Ahriz and Kit-Ying Hui

School of Computing,
The Robert Gordon University,
Aberdeen, United Kingdom
{dl, ia, ha, khui} @ comp.rgu.ac.uk

Abstract. We present a hybrid approach to Distributed Constraint Satisfaction which combines incomplete, fast, penalty-based local search with complete, slower systematic search. Thus, we propose the hybrid algorithm PenDHyb where the distributed local search algorithm DisPeL is run for a very small amount of time in order to learn about the difficult areas of the problem from the penalty counts imposed during its problem-solving. This knowledge is then used to guide the systematic search algorithm SynCBJ. Extensive empirical results in several problem classes indicate that PenDHyb is effective for large problems.

Key words: Constraint Satisfaction, Distributed AI, Hybrid Systems.

1 Introduction

Constraint satisfaction is an increasingly important paradigm for the resolution of combinatorial problems. A Constraint Satisfaction Problem (CSP) [1] is a triple (X, D, C) where $X = \{x_1, ..., x_n\}$ is a set of variables, $D = \{D_1, ..., D_n\}$ is a set of domains and C is a set of constraints which restrict the values that variables can take simultaneously. A solution to a CSP is an assignment of values to variables which satisfies all constraints.

A Distributed Constraint Satisfaction Problem (DisCSP) [2], is a CSP where the problem (variables, domains and constraints) is distributed amongst a number of agents each of which has only a partial view of the problem due to privacy issues, communication costs or security concerns. Thus, in order to solve the problem, agents must communicate and cooperate whilst disclosing as little information as possible. Assumptions commonly shared by researchers in the field [2] are: (i) message delays are finite and for any pair of agents, messages are received in the order that they are sent; (ii) each agent is responsible for one variable.

Existing solution methods for DisCSPs can be classified as systematic search and local search methods. In systematic algorithms, ordered agents sequentially instantiate their variable, backtracking to the previous agent if no consistent value is found. Zivan and Meisels [3] devised SynCBJ, a distributed version of conflict-directed backjumping [4] combined with dynamic backtracking [5]. Systematic search algorithms are complete[1].

[1] They either find a solution to a problem or detect that the problem is unsolvable.

Consequently, distributed local search algorithms have been devised which, for large problems, converge quicker to a solution but are generally incomplete. Distributed local search approaches iteratively improve an initial set of values until a set of values which is a solution is found. However, it may find one set of non-optimal values (local optima) always appears more promising than moving to other combinations of values (the neighbourhood) and get stuck. Therefore, local search approaches rely heavily on strategies for escaping local optima, e.g. weights on constraints [2] or penalties on values [6]. For large problems, they are faster than the systematic approaches.

There are only two distributed hybrid approaches which combine both types of search to produce hybrid algorithms which are 'fast' and complete. DisBOBT [7] uses Distributed Breakout [2] as its main problem-solver and, if Distributed Breakout fails to solve the problem, its weight information orders the agents for an SBT [3] search. LSDPOP [8] is an optimisation algorithm running the systematic algorithm DPOP [9], until the maximum inference limit is exceeded when local search guided by DPOP is run to find the best solution to the problem.

We introduce a hybrid approach, PenDHyb, combining penalty-based local search, DisPeL [6], with systematic search, SynCBJ, for distributed constraint satisfaction.

The remainder of this paper is structured as follows. Our hybrid approach, PenDHyb, is explained in Section 2; Section 3 presents empirical results on both solvable and unsolvable problems. Finally, Section 4 concludes the paper.

2 PenDHyb: Penalty-based Distributed Hybrid Algorithm

We propose a new approach, PenDHyb, for Distributed Constraint Satisfaction which combines penalty-driven local search (DisPeL) with systematic search (SynCBJ) in order to speed-up the latter. In the former type of search, when a quasi-local optimum is reached, penalties are imposed on 'current' variable values causing constraint violations. Penalties therefore indicate values that, though looking promising, fail to lead to a solution. The higher the penalties accumulated by a value, the less desirable it becomes. The penalty and value information learnt during penalty-driven problem solving can be used to guide systematic search as follows:

- **Difficult variables:** Penalties on values are used to learn which variables are difficult to assign during problem solving. A variable which has many heavily penalised values is seen as more troublesome than a variable whose values have few or no penalties. Variables are ordered according to their degree and difficulty (penalties) and this order is used to drive the systematic search algorithm.
- **Best variable values:** the best solution found (the one with the least constraint violations) in the penalty-based algorithm, is used for value ordering in the systematic search.

DisPeL [6] is an iterative improvement algorithm where agents take turns to improve a random initialisation in a fixed order. In order to resolve deadlocks (quasi-local-optima where an agent's *view* remains unchanged for 2 iterations), DisPeL applies penalties to variable values which are used in a 2-phased strategy as follows: (i) First the values are penalised with a *temporary penalty* in order to encourage agents to assign

other values and; (ii) If the temporary penalties fail to resolve a deadlock *incremental penalties* are imposed on the culprit values. In the more efficient Stoch-DisPeL [6], agents decide randomly to either impose a temporary penalty or to increase the incremental penalty. In the remainder of this paper we refer to Stoch-DisPeL as DisPeL.

SynCBJ is a synchronous systematic search algorithm where each agent keeps track of the reasons why values have been eliminated from their variable's domain. When a bactrack step is required, the agent is able to determine the variable responsible for the conflict and backjumps to the agent holding that variable. This increases the performance of the algorithm very substantially when compared to SBT.

In order to learn penalty information we modified DisPeL by adding:

- A *penalty counter* pc_i for each variable v_i. pc_i is incremented whenever a penalty is imposed on any of v_i's values. Unlike penalties on values, penalty counters are never reset and, therefore, highlight repeated penalisation of variables, i.e. *troublesome* variables.
- A *best value* bv_i store for each variable v_i which keeps the value participating in the best solution found by DisPeL so far [2].

Our hybrid algorithm, PenDHyb, combines DisPeL and SynCBJ. After agent initialisation, a standard DisPeL search runs (as described for Stoch-DisPeL in [6]) but only for a very small number of cycles e.g. 21 cycles for randomly generated problems with 40 variables. We ran a series of experiments and determined that a very small number which steadily increases as the number of variables increase was optimal. If the problem is solved, the solution is returned. Otherwise, variables are arranged, in descending order, according to their degree and penalty count before SynCBJ is run. In addition to the variable ordering information, SynCBJ makes use of value ordering information as follows: for each variable v_i, the first value to be tried is the best value bv_i found by DisPeL, i.e. the one participating in the best instantiation found.

PenDHyb is complete since either DisPeL reports a solution within the small number of cycles (typically DisPeL solves 5% of problems) or SynCBJ runs. Since SynCBJ is complete, completeness of PenDHyb is guaranteed. PenDHyb is sound since both DisPeL and SynCBJ are sound [6, 3].

3 Empirical Evaluation

Our SynCBJ implementation was verified with the distributed randomly generated problems described in [3] with the results at least as good as those reported by SynCBJ's authors. We use SynCBJ with max-degree variable ordering which obtains substantially better results than lexicographic ordering.

We evaluated PenDHyb on solvable and unsolvable distributed randomly generated problems measuring the two established costs for DisCSPs: (i) the number of messages sent; (ii) the number of constraint checks performed. Although CPU time is not an established measure for DisCSPs [10], we also measured it and the results obtained were consistent with the other measures used.

[2] Note that determining the best solution does not incur any additional messages.

Table 1. Performance of SynCBJ and PenDHyb on randomly generated problems.

N. messages	**solvable problems**				**unsolvable problems**			
n	**30**	**40**	**50**	**60**	**30**	**40**	**50**	**60**
SynCBJ	2301	22590	262178	1897645	*5307*	55692	557359	3069301
PenDHyb	*2115*	*18566*	*100417*	*486301*	5546	*51916*	*451218*	*2686295*
N. cnstr. checks	**solvable problems**				**unsolvable problems**			
n	**30**	**40**	**50**	**60**	**30**	**40**	**50**	**60**
SynCBJ	*11489*	119209	1344941	10421510	*24790*	285591	2924331	17153383
PenDHyb	12041	*116573*	*714961*	*2975784*	27913	*284847*	*2424052*	*15077229*

Table 2. SynCBJ and PenDHyb on graph colouring problems for $degree = 5$.

N. messages	**solvable problems**				**unsolvable problems**			
n	**125**	**150**	**175**	**200**	**125**	**150**	**175**	**200**
SynCBJ	18781	75778	191988	722256	127054	660334	1957622	6793331
PenDHyb	*18577*	*60005*	*161213*	*463601*	*113590*	*557434*	*1849564*	*5357801*
N. cnstr. checks	**solvable problems**				**unsolvable problems**			
n	**125**	**150**	**175**	**200**	**125**	**150**	**175**	**200**
SynCBJ	*46234*	178942	477713	1750199	309383	1587410	4518670	15694031
PenDHyb	52534	*162748*	*416520*	*463601*	*281142*	*1327274*	*4498886*	*12527968*

We evaluated PenDHyb against SynCBJ on a wide variety of randomly generated problems ($n \in \{30, 40, 50, 60\}$, $d \in \{8, 9, 10, 11, 12\}$, $p1 \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$ and $p2 \in \{0.1, 0.15, ..., 0.95\}$) where n is the number of variables, d is the domain size, $p1$ is the constraint density and $p2$ is the constraint tightness. We present the results at the phase transition point which represents hard problems for SynCBJ. Other tightness values showed similar performance for both algorithms. The results, shown in Table 1 for problems with ($n \in \{30, 40, 50, 60\}$, $d = 10$, $p1 = 0.15$ and $p2 = 0.6(30), 0.5(40), 0.45(50), 0.4(60)$), are median values over 100 problems. For solvable problems, PenDHyb is significantly more efficient than SynCBJ with performance difference increasing with the number of variables. For unsolvable problems SynCBJ is marginally better on problems with 30 variables but PenDHyb is substantially better on problems with 40 or more variables.

We also evaluated the performance of PenDHyb against SynCBJ on distributed graph colouring problems ($nodes \in \{125, 150, 175, 200\}$, domain size $d = 3$ and degree $k \in \{4.6, ..., 5.3\}$). These problems are of similar size to the ones used for the experiment on randomly generated problems above. Median values over 100 solvable problems and 100 unsolvable problems are shown in Table 2 for problems with a degree of 5. The results showed that PenDHyb is significantly more efficient for both solvable and unsolvable problems. This efficiency becomes more profound as the number of nodes increase and thereby mirrors the performance of PenDHyb with randomly generated problems namely that PenDHyb is substantially more efficient on medium to large-sized problems. Experiments for other degrees (not shown here) gave similar results, i.e. PenDHyb performed better, especially for graphs with a large number of nodes.

4 Conclusions

We have presented, PenDHyb, a hybrid approach to Distributed Constraint Satisfaction using penalty-based local search algorithm, DisPeL, to learn about the problem with the knowledge gained guiding a systematic search algorithm, SynCBJ.

We also evaluated other methods of exploiting the knowledge gained from running DisPeL to provide variable and value ordering for SynCBJ. We found that the best performing method was the one used in PenDHyb, where variables are sorted using max-degree and penalties and values are prioritised using sticking values.

We have shown that PenDHyb's performance is significantly better than systematic search for large, difficult problems in two problem classes, randomly-generated problems and graph colouring problems.

Our future work with PenDHyb will investigate the effectiveness of our approach on non-binary problems and the approach's applicability to coarse-grained problems, where agents are responsible for more than one variable.

References

1. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier (2006)
2. Yokoo, M., Hirayama, K.: Algorithms for Distributed Constraint Satisfaction: A Review. Autonomous Agents and Multi-Agent Systems **3**(2) (2000) 185–207
3. Zivan, R., Meisels, A.: Synchronous vs asynchronous search on DisCSPs. In: Proceedings of the First European Workshop on Multi-Agent Systems (EUMA), Oxford (December 2003)
4. Prosser, P.: Hybrid Algorithms for the Constraint Satisfaction Problem. Computational Intelligence **9**(3) (1993) 268–299
5. Ginsberg, M.L.: Dynamic backtracking. Journal of Artificial Intelligence Research **1** (1993) 25–46
6. Basharu, M., Arana, I., Ahriz, H.: Stoch-DisPeL: Exploiting randomisation in DisPeL. In: Proceedings of 7th International Workshop on Distributed Constraint Reasoning, Hakodate, Japan (May 2006)
7. Eisenberg, C.: Distributed Constraint Satisfaction for Coordinating and Integrating a Large-Scale Heterogeneous Enterprise. PhD thesis, Ecole Polytechnique Federale De Lausanne (2003)
8. Petcu, A., Faltings, B.: A hybrid of inference and local search for distributed combinatorial optimization. In: Proceedings of 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IEEE Computer Society (2007) 342–348
9. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, Scotland (August 2005)
10. Meisels, A., Kaplansky, E., Razgon, I., Zivan, R.: Comparing performance of distributed constraints processing algorithms. In: Proceedings of the AAMAS-2002 Workshop on Distributed Constraint Reasoning, Bologna (July 2002) 86–93