Multi-Hyb: A Hybrid Algorithm for Solving DisCSPs with Complex Local Problems

David Lee, Inés Arana, Hatem Ahriz, and Kit-Ying Hui
School of Computing
Robert Gordon University, Aberdeen, AB25 1HG, Scotland, UK
Email: {dl, ia, ha, khui}@comp.rgu.ac.uk

Abstract

A coarse-grained Distributed Constraint Satisfaction Problem (DisCSP) is a constraint problem where several agents, each responsible for solving one part (a complex local problem), cooperate to determine an overall solution. Thus, agents solve the overall problem by finding a solution to their complex local problem which is compatible with the solutions proposed by other agents for their own local problems. Several approaches to solving DisCSPs have been devised and can be classified as systematic search and local search techniques. We present Multi-Hyb, a two-phase hybrid algorithm for solving coarse-grained DisCSPs which uses both systematic and local search during problem solving. Phase 1 generates key partial solutions to the global problem using systematic search. Concurrently, a penalty-based local search algorithm attempts to find a global solution to the problem using these partial solutions. If a global solution is not found in phase 1, the information learnt from phase 1 is used to inform the search carried out during the next phase. Phase two runs a systematic search algorithm on complex variables guided by the following knowledge obtained in phase 1: (i) partial solutions and ; (ii) complex local problems which appear more difficult to satisfy. Experimental evaluation demonstrates that Multi-Hyb is competitive in several problem classes in terms of: (i) the communication cost and (ii) the computational effort needed.

1. Introduction

A Constraint Satisfaction Problem (CSP) is a problem which can be represented by a set of variables, a corresponding set of domains (one per variable) and a set of constraints which restricts the values that variables can take simultaneously. A solution to a CSP is an assignment of a value for each variable which satisfies all constraints. Two main classes of algorithms are used for the resolution of CSPs: (i) Systematic search algorithms, which ensure completeness but can be slow and; (ii) Local search algorithms, which are incomplete, but can be, for large problems, faster than systematic algorithms [1].

Distributed Constraint Satisfaction Problems (DisCSPs) [2] are CSPs where each triplet *[variable, its domain, the constraints it is involved in]* is assigned to an agent, who is responsible for finding a consistent value to that variable. Agents only know about their own variables, the constraints they are involved and the current value for variables related (via constraints) to its own variables.

DisCSPs are frequently naturally coarse-grained, i.e. they consist of a set of inter-related sub-problems (complex local problems), each of which can be represented by a CSP with a set of variables, a corresponding set of domains and a set of constraints between variables (intra-agent constraints). Complex local problems are linked together by a set of constraints which relate variables in two or more local problems (inter-agent constraints). Specifically we concentrate on naturally distributed problems, i.e. those where the number of inter-agent constraints is substantially smaller than the number of intra-agent constraints.

Systematic search algorithms for coarse-grained DisCSPs have been proposed based on Asynchronous Backtracking (ABT) [2] and Asynchronous Weak Commitment Search (AWCS) [2]. Multi-AWCS [3] uses a local AWCS solver to ensure the satisfaction of intra-agent constraints, whilst a global AWCS solver ensures the satisfaction of inter-agent constraints. Multi-ABT [4] is a comparable extension to coarse-grained DisCSPs for ABT. Whilst these algorithms are complete, they can take exponential time and, in the case of Multi-AWCS, exponential nogood storage.

Distributed local search approaches for DisCSPs with complex local problems include DisBO-wd [5] which attaches weights to constraints with a decay mechanism and Multi-DisPeL [5] which assigns penalties to values to escape quasi-local-optima.

Whilst distributed hybrid approaches exist for one variable per agent (e.g. PenDHyb [6]), to the best of our knowledge, there are no hybrid approaches specifically designed for solving DisCSPs with complex local problems.

We present Multi-Hyb, a distributed hybrid algorithm for coarse-grained DisCSPs which combines systematic and local search techniques [7]. Multi-Hyb uses a two-phase strategy: (i) In the first phase, it concurrently learns consistent combinations of values for each complex local problem using systematic search while, at the same time, attempting to find a global solution using local search. (ii) If the first phase does not solve the problem, knowledge learnt during both systematic and local search in the first phase is used to guide a systematic search algorithm on complex variables.

2. The Multi-Hyb Algorithm

Multi-Hyb (see algorithm 1) is a novel two-phase complete distributed hybrid algorithm for solving DisCSPs with complex local problems which are naturally distributed, i.e. with a high intra-agent to inter-agent constraint ratio. In order to explain each phase and the interaction between the two phases a simple scheduling DisCSP (see figure 1) with complex local problems is used. A diagram of the two phases of Multi-Hyb is shown in figure 2.

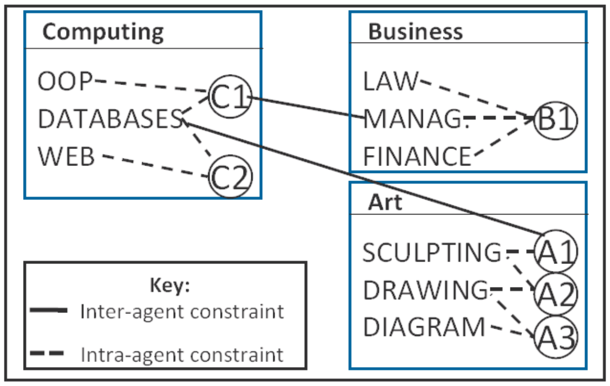

Fig. 1. A scheduling DisCSP with complex local problems.

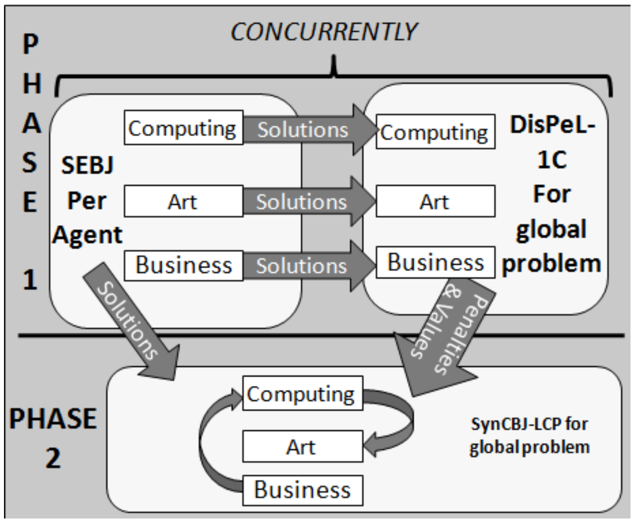

Fig. 2. The two phases of Multi-Hyb.

A University has three schools: Computing, Business and Art. Each school has a number of courses and teaches a number of modules (variables) which can either be taught by the school or by other schools (external modules). Two modules sharing a common course cannot be scheduled at the same time. When preparing their individual timetable, schools must consider their internal modules (i.e. internal constraints) as well as ensure there are no clashes with external modules (i.e. external constraints). For simplicity, classes can be scheduled at 9am, 10am, 11am and 12noon on Mondays.

Phase 1

In Phase 1, each agent finds all 'relevant' (non-interchangeable) solutions to its complex local problem using the Synchronous Exhaustive Backjumping (SEBJ) algorithm (see below). While agents are concurrently searching for local solutions with SEBJ, a DisPeL-1C (see below) search attempts to find a solution to the global problem. Phase 1 finishes when: (i) an agent detects no solution to its complex local problem and, therefore, the overall problem is unsolvable; (ii) all agents have found all 'relevant' local solutions using SEBJ or; (iii) DisPeL-1C finds a solution to the global problem. If a solution is not found, and no unsolvability has been detected, Multi-Hyb starts Phase 2 which runs the SynCBJ-CLP algorithm.

Algorithm 1 Multi-Hyb

```
1: Initialise all agents with their subproblem.
2: Run SEBJ and dynamically pass subproblems solutions to
   DisPeL-1C
3: CONCURRENTLY run DisPeL-1C
4: if SEBJ found solutions and DisPeL-1C did not find a global
   solution then
5:    Run SynCBJ-CLP algorithm
6: end if
```

SEBJ (see Algorithm 2) is a systematic search algorithm which finds all non-interchangeable solutions to a complex local problem, i.e. the set of all solutions which differ on at least one value for an external variable (a variable linked to another complex local problem).

Algorithm 2 SEBJ

```
1: initialise - order external variables before internal variables
2: Set $solutionFound \leftarrow$ false and $solutionCount \leftarrow 0$
3: for each variable $v_i$ do
4:    for each value $d_i$ in variable $v_i$'s domain do
5:       if all higher priority constraints are satisfied then
6:          if $solutionFound =$ true OR $solutionFound =$ false
            AND all higher priority nogoods are not consistent with
            all variable values then
7:             assign value $d_i$ to variable $v_i$
8:             move to next variable in for loop.
9:          end if
10:      else if $solutionFound =$ false then
11:         for each higher priority constraint which is violated do
12:            Add the variable/value pair to a nogood for value $d_i$
               to variable $v_i$
13:         end for
14:      end if
15:   end for
16:   if variable $v_i$ has no assigned value then
17:      if first variable is $v_i$ and $solutionCount = 0$ then
18:         return "unsolvable problem"'
19:      else if first variable is $v_i$ then
20:         return "all solutions found"
21:      else if $solutionFound =$ false then
22:         Create variable $v_i$'s conflict set with all variables in-
            volved in nogoods for values of variable $v_i$
23:         Backjump to lowest priority variable in the conflict set.
24:      else if $solutionFound =$ true then
25:         Backtrack to previous variable in for loop.
26:         if lowest priority variable is first variable then
27:            set $solutionFound \leftarrow$ false.
28:         end if
29:      end if
30:   end if
31: end for
32: Add solution to solution store.
33: Set $solutionFound \leftarrow$ true and increment $solutionCount$
34: Restart for loop with variable $v_i$ as last variable which has
    external links with value $d_i$ as the next value in its domain.
```

SEBJ is similar to the local solver algorithm with interchangibility solutions presented in [8] with the following differences: (i) SEBJ uses conflict-directed backjumping to determine all possible solutions; (ii) SEBJ runs concurrently with DisPeL-1C once at least one solution to each local complex problem is found. SEBJ is sound and complete with regard to identifying all solutions of external relevance.

SEBJ terminates since it is a systematic algorithm run in a centralised environment so each instance of SEBJ terminates when either: (i) it has found all non-interchangeable solutions to its local problem; (ii) it finds that it has no solution to its local problem and has informed all other agents; (iii) one of the agents sends a message stating that the problem is unsolvable; (iv) DisPeL-1C sends a message stating that it has found a solution.

DisPeL-1C: The DisPeL local search algorithm [9] uses a penalty-based approach to escaping local minima which is further improved in [10][1]. DisPeL-1C substantially differs from DisPeL as follows; (i) DisPeL-1C continuously imposes penalties when values are inconsistent without waiting until a quasi-local-minimum is detected; (ii) DisPeL-1C's variables are complex, each representing all externally relevant variables for a complex local problem; (iii) In DisPeL-1C variable values are dynamically added to their domain (as SEBJ finds them); consequently, DisPeL-1C could solve a problem without knowing all solutions to the local problems that SEBJ instances will generate; (iv) DisPeL-1C keeps track of the best solution (with fewest contraint violations) found so far; (v) DisPeL-1C only considers the inter-agent constraints since the intra-agent constraints have already been checked by SEBJ. DisPeL-1C may discover that the subproblem's solutions can be extended to form a global solution to the problem. If this is the case, Multi-Hyb terminates. Otherwise, if one instance of SEBJ terminates with no solutions for a complex local problem then both DisPeL-1C and Multi-Hyb terminate. However, if all SEBJ instances find at least one solution and then terminate, DisPeL-1C terminates and Multi-Hyb moves to phase 2.

Phase 2

Phase two runs a single algorithm, SynCBJ-CLP.

SynCBJ-CLP: The SynCBJ algorithm [11] for complex local problems (SynCBJ-CLP) uses one complex variable per agent, with each variable representing all the externally relevant variables of a complex local problem. The algorithm explores partial solutions generated by SEBJ such as $(OOP = 9am, Databases = 10am)$ and $(Manag = 10am)$ to see if they extend to a global solution. Hence, SynCBJ-CLP only considers the inter-agent constraints (for example $OOP \neq Manag$) and ignores the intra-agent constraints, since these have already been checked by SEBJ. Knowledge sharing inspired by PenDHyb [6] exists so that SynCBJ-CLP uses the following knowledge learnt by DisPeL-1C: (i) difficult areas of the problem and; (ii) best 'solution' found so far learnt by DisPeL-1C. Thus, the penalties incurred by a variable's values during the running of DisPeL-1C indicate the comparative level of difficulty the algorithm has had in finding a consistent value for that variable. SynCBJ-CLP uses a reordering scheme combining a weight of 70% for the sum of DisPeL-1C's incremental penalties on variable values (which is periodically reset) with a weight of 30% of the cumulative penalty count of all penalties imposed by DisPeL-1C on a variable and max-degree ordering so that agents with 'difficult' variables have a higher priority. In addition, the variable values involved in the best 'solution' found by DisPeL-1C are tried first (value ordering).

SynCBJ-CLP is efficient through its use of: (i) complex variables, aggregating all variables of the agent's complex local problem thereby having one complex variable per agent; (ii) only inter-agent constraints are given consideration (the same constraints considered by DisPeL-1C). Since SynCBJ is complete and variations introduced only change the ordering of agents and first variable value, this phase is complete. Since this phase will run if DisPeL-1C is unable to find a solution, the Multi-Hyb algorithm is complete.

3. Experimental Evaluation

We compared Multi-Hyb with Multi-AWCS, Multi-DisPeL and DisBO-wd on distributed randomly generated problems, distributed graph colouring problems and distributed scheduling problems. We measured: (i) the number of Non-Concurrent Constraint Checks (NCCCs) performed; (ii) the number of messages sent and; (iii) percentage of problems solved (only Multi-DisPeL and DisBO-wd as they are incomplete algorithms). For Multi-DisPeL and DisBO-wd, we use a cut-off of $100n$ iterations (where n is the number of variables) and $200n$ iterations respectively (since 2 DisBO-wd cycles of *improve* and *ok?* equal one Multi-DisPeL cycle).

Extensive empirical evaluations varied the number of variables (60-200), the domain sizes (5-10), constraint tightness (0.35-0.5), constraint densities (0.15-0.2) and number of agents (5-25). We considered naturally distributed DisCSPs with complex local problems having 70% to 90% intra-agent constraints with the remainder being inter-agent constraints. For each problem type, 100 different problems were solved and average and median results calculated.

Median results for distributed randomly generated problems appear in Table 1 with $n \in \{60, ..., 175\}$, 5 agents, 8 domain values, 0.2 constraint density and 0.35 constraint tightness. Bold indicates the best performing algorithm. * indicates that the algorithm could not solve all problems. In these cases, the effort wasted (number of NCCCs and number of messages) was not included in the results.

For larger randomly generated problems (80 variables and above), Multi-Hyb gives the best results. There are a few occasions where Multi-DisPeL uses slightly less messages but the difference is very small when compared with the large difference in the number of NCCCs. For smaller problems, Multi-AWCS is best for NCCCs but uses substantially more

1. In the remainder of this paper, we refer to this latter version as DisPeL.

TABLE 1. Results for solvable random problems

		Median n. of messages			
n. vars	% intra-:inter agent constr.	Multi -Hyb	Multi -AWCS	Multi -DisPeL	DisBO -WD
60	90:10	**399**	4834	536	1150*
60	80:20	**197**	5287	422	1165
60	70:30	818	4475	**496**	985
80	80:20	143	3991	**104**	335
80	70:30	**89**	6076	108	295
100	80:20	**56**	5922	**56**	235
100	70:30	78	7235	**60**	225
125	80:20	**20**	6297	40	225
125	70:30	60	9218	**40**	205
150	80:20	**20**	6803	28	215
150	70:30	**30**	14554	32	195
175	80:20	**20**	10707	24	210
175	70:30	**20**	15126	24	190
		Median n. of NCCCs			
n. vars	% intra-:inter agent constr.	Multi -Hyb	Multi -AWCS	Multi -DisPeL	DisBO -WD
60	90:10	**163585**	165118	1187335	469162*
60	80:20	277408	**194432**	949616	440862
60	70:30	2761171	**182936**	1148704	353862
80	80:20	**118874**	149599	588111	283827
80	70:30	**169884**	265274	606084	262707
100	80:20	**107836**	285431	690977	339423
100	70:30	**132031**	385969	690455	324668
125	80:20	**106435**	357508	952787	509090
125	70:30	**125553**	600688	936775	485739
150	80:20	**100020**	441287	1362161	728427
150	70:30	**120105**	1302570	1281866	682116
175	80:20	**98875**	885339	1926771	976712
175	70:30	**110325**	1453996	1831216	908710

messages. There is only one occasion (60 variables, 70:30 intra-agent to inter-agent constraints) where Multi-Hyb does not give the best results for either messages or NCCCs - this is because 60-variable problems are fairly small.

We also compared Multi-Hyb against Multi-AWCS on unsolvable distributed randomly generated problems. A comparison with Multi-DisPeL and DisBO-wd for unsolvable problems is not possible since these algorithms are incomplete. Multi-Hyb quickly determined that the problem was over-constrained, particularly for problems with an inconsistent complex local problem. Multi-AWCS used far more messages and NCCCs. The results are not shown owing to space limitations.

We conducted experiments on distributed graph colouring problems with 150 and 200 nodes with 15 to 25 agents, 3 colours and degree $\in \{4.9, 5.1\}$. The ratio of intra-agent to inter-agent constraints was $85 : 15$ and $90 : 10$. For solvable problems, Multi-Hyb was always best in terms of messages and whilst occasionally Multi-AWCS used slightly less NCCCs, Multi-AWCS used far more messages. For unsolvable problems, Multi-Hyb outperformed Multi-AWCS by several orders of magnitude for both messages and NCCCs. We also ran tests on distributed scheduling problems based on Brito's generator [12]. Our problems had 50-80 meetings, 5 departments (agents), a timeframe of 6 or 7 time units and a 0.18 constraint density. The ratio of intra-agent to inter-agent constraints was $85 : 15$ and $90 : 10$. Departments with common meetings have a random distance $\in \{1, 2, 3\}$ time units. For most solvable problems, Multi-Hyb performed best for both number of messages and NCCCs. For unsolvable problems, Multi-Hyb outperformed Multi-AWCS by several orders of magnitude. These results are not shown owing to space limitations.

4. Conclusions

We have presented Multi-Hyb, a concurrent distributed complete algorithm for solving DisCSPs with complex local problems. Multi-Hyb uses SEBJ to concurrently find all externally relevant solutions for each agent's local problem whilst also running DisPeL-1C to find a suitable combination of local problem solutions which does not violate any inter-agent constraint. If SEBJ does not detect unsolvability and has found all non-interchangeable solutions and DisPeL-1C does not find a solution to the problem, SynCBJ-CLP algorithm is run. This algorithm benefits from: (i) not having to check intra-agent constraints since it has the local problem solutions obtained by SEBJ; (ii) being able to use possible 'best values' found by DisPeL-1C; (iii) using the penalty information gained by DisPeL-1C to determine 'difficult local problems' and hence, reorder agents. Multi-Hyb generally significantly outperforms Multi-AWCS, Multi-DisPeL and DisBO-wd often by several orders of magnitude.

References

[1] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*. Elsevier, 2006.

[2] M. Yokoo and K. Hirayama, "Algorithms for Distributed Constraint Satisfaction: A Review," *Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 2, pp. 185–207, 2000.

[3] M. Yokoo and K. Hirayama, "Distributed Constraint Satisfaction Algorithm for Complex Local Problems." in *ICMAS*, 1998, pp. 372–379.

[4] K. Hirayama, M. Yokoo, and K. Sycara, "An Easy-Hard-Easy Cost Profile in Distributed Constraint Satisfaction," *Transactions of Information Processing Society of Japan*, vol. 45, pp. 2217–2225, 2004.

[5] M. Basharu, I. Arana, and H. Ahriz, "Solving Coarse-grained DisCSPs with Multi-DisPeL and DisBO-wd," in *Proceedings of 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*. IEEE Press, 2007, pp. 335–341.

[6] D. Lee, I. Arana, H. Ahriz, and K.-Y. Hui, "A Hybrid Approach to Distributed Constraint Satisfaction," in *Artificial Intelligence: Methodology, Systems,and Applications. 13th International Conference, AIMSA 2008 Proceedings*, D. Dochev, M. Pistore, and P. Traverso, Eds., Varna, Bulgaria, September 2008, pp. 375–379.

[7] D. Lee, I. Arana, H. Ahriz, and K.-Y. Hui, "A Hybrid Approach to Solving Coarse-grained DisCSPs," in *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, May 2009, pp. 1235–1236.

[8] D. Burke, "Exploiting Problem Structure in Distributed Constraint Optimisation with Complex Local Problems," Ph.D. dissertation, National University of Ireland, Cork, 2008.

[9] M. Basharu, I. Arana, and H. Ahriz, "Solving DisCSPs with Penalty Driven Search," in *Proceedings of AAAI 2005 - The Twentieth National Conference of Artificial Intelligence*, Pittsburgh, Pennsylvania, July 2005, pp. 47–52.

[10] M. Basharu, I. Arana, and H. Ahriz, "StochDisPeL: Exploiting Randomisation in DisPeL," in *Proceedings of 7th International Workshop on Distributed Constraint Reasoning*, Hakodate, Japan, May 2006, pp. 117–132.

[11] R. Zivan and A. Meisels, "Synchronous vs Asynchronous search on DisCSPs," in *Proceedings of the First European Workshop on Multi-Agent Systems (EUMA)*, Oxford, December 2003.

[12] I. Brito, "Distributed Constraint Satisfaction," Ph.D. dissertation, Institut d'Investigacio en Intel.ligencia Artificial Consejo Superior de Investigaciones Cientificas, 2007.