

Approximations in Distributed Optimization

Adrian Petcu and Boi Faltings

Ecole Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne (Switzerland)
{adrian.petcu, boi.faltings}@epfl.ch

Abstract. We present a parameterized approximation scheme for distributed combinatorial optimization problems based on dynamic programming. The algorithm is a utility propagation method and requires a linear number of messages. For exact computation, the size of the largest message is exponential in the width of the constraint graph. We present a distributed approximation scheme where the size of the largest message can be adapted to the desired approximation ratio, α . The process is similar to a distributed version of the minibucket elimination scheme, performed on a DFS traversal of the problem.

The second part of this paper presents an anytime version of the algorithm, that is suitable for very large, distributed problems, where the propagations may take too long to complete.

Simulation results show that these algorithms are a viable approach to real world, loose optimization problems, possibly of unbounded size.

1 Introduction

Constraint satisfaction and optimization are powerful paradigms that model a large range of tasks like scheduling, planning, optimal process control, etc.

To address distributed optimization, complete algorithms like OptAPO, ADOPT and DPOP have been recently introduced.

In distributed systems, in addition to computational costs, one has to take into account the communication overhead incurred as a consequence of the message exchange. Backtracking algorithms like ADOPT [3] work by trying out many combinations of value assignments, and each one of these state changes requires at least a message. This translates into an exponential amount of single-value messages, which generally entails a big communication overhead that should be avoided.

Centralized/distributed hybrids like OptAPO [2] mitigate the communication explosion by centralizing parts of the problem in some agents, and solving these parts centrally, and then distributing the results. Arguably, this approach suffers from privacy problems, and performance bottlenecks in the centralizing nodes.

Dynamic programming algorithms like *DPOP* [4] generate a linear number of messages. However, in case the problems have high induced width, the messages generated in the high-width areas of the problem get large.

We propose in this paper ADPOP, an approximate version of DPOP, which allows the desired tradeoff between solution quality and computational complexity (see section 3). The second part of this paper (section 4) presents AnyPOP, an anytime version of ADPOP, which provides increasingly accurate solutions while the propagation is

still in progress. This makes it suitable for very large, distributed problems, where the propagations may take a long time to complete.

Simulation results on distributed meeting scheduling problems show this approach to be viable for real world, loose, possibly unbounded optimization problems.

2 Definitions & notation

A discrete *multiagent constraint optimization problem* (MCOP) is a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ such that:

$\mathcal{X} = \{X_1, \dots, X_m\}$ is the set of *variables/solving agents*; $\mathcal{D} = \{d_1, \dots, d_m\}$ is a set of *domains* of the variables, each given as a finite set of possible values. $\mathcal{R} = \{r_1, \dots, r_p\}$ is a set of *relations*, where a relation r_i is a function $d_{i1} \times \dots \times d_{ik} \rightarrow \mathbb{R}$ which denotes how much utility is assigned to each possible combination of values of the involved variables. The goal is to find an assignment \mathcal{X}^* for the variables X_i that maximizes the aggregate overall utility.

3 ADPOP - a configurable approximation method

This is an approximate version of the DPOP algorithm from [4]. DPOP is a distributed version of the bucket elimination scheme from [1], which works on a DFS ordering.

DPOP has 3 phases. First, a DFS traversal of the graph is done using a standard distributed DFS algorithm.

The second phase (*UTIL* propagation) is a bottom-up process, which starts from the leaves and propagates upwards only through tree edges. The agents send *UTIL* messages to their parents. These messages summarize the influence of the sending variable and its whole subtree on the rest of the problem. They are equivalent to the induced constraints computed in the variable elimination steps in the bucket elimination scheme.

When the *UTIL* propagation reaches the root, the top-down solution reconstruction process is initiated.

3.1 Approximations: dropping dimensions through approximate projections

The time/space complexity of DPOP's utility propagation is exponential in the induced width (see [4] for the proof). Therefore, in case the problem has high induced width, it is no longer feasible to compute and send exact messages. However, if we renounce exactness, we can impose a limit *maxDims* on the maximum number of dimensions any message in the system can carry. When the dimensionality of the outgoing message exceeds this limit, the algorithm drops a set \mathcal{S} of dimensions to stay below the limit. This is done by applying a maximal/minimal projection on the respective dimensions (retains the upper/lower bounds w.r.t. the respective variables). The set \mathcal{S} of dimensions to be dropped can be selected according to a greedy process. The two resulting messages are bundled together and sent to the parent as upper/lower bounds.

This process is similar to Dechter's minibucket elimination scheme (see [1]). However, notice that *ADPOP* is a distributed algorithm, with a well-defined elimination order, given by a DFS traversal of the problem graph. This particular ordering is very well

suited for a distributed setting for a number of reasons. First, it can easily be combined with the *most constrained node* heuristic to obtain low-width orderings. Second, it ensures a good degree of parallelism since nodes in disjoint branches can work in parallel. Third, deciding how to combine incoming messages, deciding for which ones to wait and where to send the outgoing ones is made straightforward by the DFS hierarchy.

Furthermore, when the *maxDims* bound is exceeded, then some dimensions are forcibly removed by approximate projections, as opposed to computing several lower dimensionality messages, as the minibucket scheme does.

Propagating both lower and upper bound messages gives us the ability to determine locally the maximal distance δ from the optimal solution for each value from the domain of the current variable.

This enables several ways of reasoning with bounds. First, let us consider the *UTIL* propagation. If a specified approximation ratio α is given, we can make sure that our future solution will observe this bound by dropping only as many dimensions as allowed by it. Alternatively, if a *maxDims* bound is specified, then we can drop as many dimensions as needed, and still compute an overall δ that shows how far from the optimum is the solution. In case the obtained δ is not satisfactory, one can repeat the process, with increased *maxDims*, reusing the work that was previously done in the areas where *maxDims* was not exceeded.

Second, during the value assignment propagation, one can choose the assignments according to two strategies. An *optimistic strategy* that assigns the values with the highest upper bounds gives some chances of finding good solutions by choosing "promising" assignments. On the other hand, a *pessimistic strategy* that chooses the values with the highest lower bounds offers a guarantee on the quality of the chosen solution.

Algorithm complexity In all cases, this algorithm produces a linear number of messages. Its complexity lies in the size of the *UTIL* messages.

The worst case is when the exact solution is required (*maxDims* = ∞ , or $\alpha = 1$). In this case, the complexity equals the induced width of the graph [4]. If the bound *maxDims* is imposed and is smaller than the width, no message larger than this is produced, and complexity is exponential in this bound.

In case an approximation ratio is specified, and *maxDims* is infinite, in the worst case complexity is again exponential in the width of the graph.

4 AnyPOP - an anytime algorithm for large optimization problems

In large, distributed constraint networks, it may take a long time until these propagations complete. In the following, we develop a way to decide quickly, *locally*, the value of each variable, based on a *limited* number of *UTIL/VALUE* messages from the neighbors. As time goes by, and the propagation spreads out, and more and more *UTIL/VALUE* messages come from the neighbors, we refine these decisions. As opposed to a local search method, we obtain *guarantees* on the quality of the solution, even before allowing the propagations to complete. There are obvious advantages to this approach: one can quickly start with a reasonably good solution, and refine it as time goes by.

The intuition is simple: the value taken by any node X_i can have an influence on the rest of the problem only through the constraints between X_i and its direct neighbors.

UTIL messages received by X_i already sum up its influence on the sending subtree. Thus, based on the set of *UTIL* messages X_i already received, and on the valuation structure of the constraints between X_i and its neighbors that did not already send *UTIL* messages, X_i can decide with a certain error bound what is the effect of each one of its values on the rest of the problem.

In some cases, when these error bounds are sufficiently low, X_i can decide on an assignment for itself even before receiving all of its *UTIL/VALUE* messages. In such a case, one can simply start the *VALUE* propagation phase immediately, without waiting for the rest of the *UTIL/VALUE* messages to come.

AnyPOP also exhibits some built-in fault tolerance. If messages are lost, solution quality degrades. However, the algorithm still provides the best solution it can infer based on the information that *was* sent/received successfully.

5 Experimental evaluation

Our experiments were performed on distributed meeting scheduling problems, where a set of agents try to jointly find the best schedule for a set of meetings. Each agent has a variable for each meeting it is involved in. The values are the possible starting times. All agents must agree on the start time of each meeting, and an agent cannot participate in 2 meetings at the same time. Each agent assigns to each meeting at each particular time a certain utility, and the task is to find the schedule that maximizes the overall utility.

We ran experiments on an especially difficult problem with 70 agents, 140 variables and 204 binary constraints. The induced width is 7, meaning that the biggest message holds over two million values. We ran the algorithm with increasing *maxDims*. Table 1 shows the results in this order: maximal dimensionality, maximal distance δ from the optimum for all *UTIL* messages, the average δ per message, the distance of the approximate solution to the true optimum, the total amount of *UTIL* information transmitted (the sum of the sizes of the individual *UTIL* messages), maximal message size, and the utility of the solutions found.

<i>maxDims</i>	Max δ /msg %	Avg δ /msg %	δ /overall %	Total UTIL payload	Max msg size	Utility
1	44.83	13.55	2.90	2104	16	2278
2	36.00	4.54	2.69	10032	128	2283
3	17.14	1.27	2.43	39600	1024	2289
4	13.11	0.57	0.81	130912	8192	2327
5	10.00	0.19	0.43	498200	65536	2336
6	1.36	0.04	0.30	1808904	524288	2339
7	0.00	0.00	0.00	3614064	2097152	2346

Table 1. Max. dimensions vs. solution accuracy: problem with 140 vars, 204 constraints,width=7

The accuracy of the solutions increases with the increase of *maxDims*, culminating with the optimal value for *maxDims* = 7. However, there is also a dramatic increase in computation effort and network load. If we compare the first and the last lines of

the table, we see that we can achieve a solution which is within 3% of the optimum with 3 orders of magnitude less effort (2k values vs. 3M). Therefore, in some cases it is beneficial to settle for a suboptimal solution obtained with much less effort.

Snapshot #	Max δ /var %	Avg δ /var %	Utility	δ /overall %	Assig changes
1	94.44	80.77	1555	33.72	0
2	66.07	16.7	1625	30.73	99
3	42.42	3.92	2036	13.21	73
4	13.51	1	2254	3.92	19
5	13.51	0.94	2289	2.43	1

Table 2. *AnyPOP* dynamic evolution: problem with 140 vars, 204 constraints,width=7

To test simultaneously both *AnyPOP*'s anytime performance and its ability to deal with low resources, we performed another experiment on the same instance, with $maxDims=3$. We took 5 runtime snapshots: the first snapshot was taken *before sending/receiving any message*, and subsequent ones after each node has received another message. The last snapshot is taken after all messages are sent/received. The assignments discovered by each of the snapshots are used to compute the overall utility. We notice a steady progress of the algorithm towards a solution, culminating with the best solution found by *AD-POP* on the same test problem, with the same bound $maxDims = 3$. There is also a steady decrease of the error bounds, and of the assignment changes between snapshots.

6 Conclusions and future work

We proposed an approximate algorithm for distributed optimization, allowing for the desired tradeoff between solution quality and computational complexity. We also presented an anytime version of this algorithm, suitable for large distributed problems. Experimental results show that these algorithms are a viable approach to large but loose real world optimization problems.

Future work includes finding heuristics for generating DFS trees with low induced width and "intelligent" selection of the dimensions to be dropped out.

References

1. Rina Dechter. Bucket elimination: A unifying framework for processing hard and soft constraints. *Constraints: An International Journal*, 7(2):51–55, 1997.
2. Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, 2004.
3. P. J. Modi, W. M. Shen, and M. Tambe. An asynchronous complete method for distributed constraint optimization. In *Proc. AAMAS*, 2003.
4. Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05*, Edinburgh, Scotland, Aug 2005.