MRPlanner: An Agent-Based MMRP Constraint Programming System*

Junwei Yang
Dept. Equip. Remanufacture Eng.
Academy of Armored Force Engineering
Beijing 100072, China
yangjunwe@sina.com

Xiaojing Huo
M&E Engineering College
Hebei Agricultural University
Baoding 071001, China
xjhuojxteng@yahoo.com.cn

Zhongxiang Hu, Xiaojun Shi
Nat. Key Lab. Remanufacturing
Academy of Armored Force Engineering
Beijing 100072, China
troopzhu@163.com

***Abstract*—Manufacturing material resource planning (MMRP) increasingly involves complex sets of objectives and constraints; therefore traditional approaches typically result in a large monolithic model that is difficult to solve, understand, and maintain. The paper presents a multiagent-based constraint programming system, namely MRPlanner, which is aimed to solving large, particularly combinatorial, problems in MMRP. The system uses an asynchronous team (A-Team) architecture in which multiple problem solving agents cooperate with each other by exchanging results to produce a set of non-dominated solutions that show the tradeoffs between objectives. The agent-based approach employed in the system significantly improves the performance, reliability, intelligence and automation of constraint programming.**

I. INTRODUCTION

Dealing with ever-increasing product complexity, agility, and speed, today's manufacturing material resource planning (MMRP) increasingly involves complex sets of objectives and constraints, and variations of uncertainty and randomicity. Traditional approaches to modeling MMRP (e.g., mathematical programming techniques) typically result in a large monolithic model that is difficult to solve, understand, and maintain.

Constraint programming is an emergent software technology for declarative description and effective solving of large, particularly combinatorial, problems especially in areas of planning and scheduling [1]. It has been successfully applied in numerous domains include operations research, database systems, electrical engineering and circuit design. Recent researches reveal that many artificial intelligence (AI) techniques have been integrated to enhance its capability to represent and automatically enforce diverse and complex constraints inherent in large, real-world applications [2].

We propose here an agent-based constraint programming system for solving various MMRP constraint satisfaction problems (CSP). The multi-agent system, namely MRPlanner, employs and extends asynchronous team (A-Team) [3], [4], an architecture in which multiple problem solving agents cooperate with each other through a population of solutions. Figure 1 gives an overview of the system, which contains the following six types of agents:

*Supported in part by grants from National Natural Science Foundation (No. 50235030 and No. 513270102) and National Grand Fundamental Research 973 Program (No. 2006BAF02A19) of China.

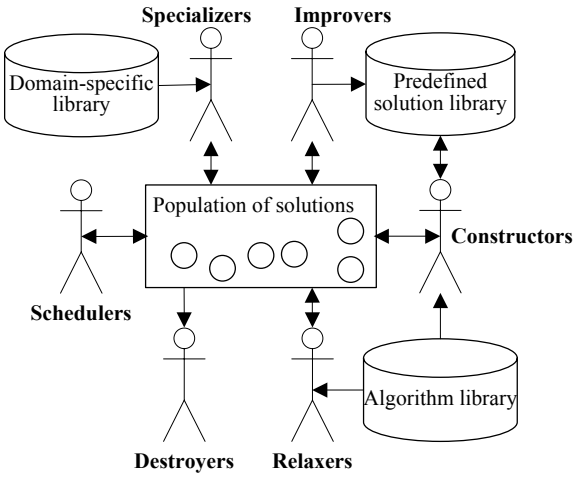

Fig. 1. The fundamental structure of MRPlanner

- *Scheduler*: the users that act agents by proposing initial problems (empty solutions) or by modifying existing solutions.
- *Specializers*: the agents that build CSP specifications for empty solutions.
- *Constructors*: the agents that select algorithms to generate initial solutions according to separated CSP specifications.
- *Improvers*: the agents that take existing solutions from the population and modify them to produce better solutions.
- *Relaxers*: the agents that take infeasible or over-complicated solutions from the population and relax some constraints for the problem.
- *Destroyers*: the agents that remove infeasible, inefficient, or redundant solutions from the population.

The remaining of the paper is structured as follows: Section 2 gives a brief introduction of CSP and A-Team architecture, sufficient to understand the paper; Section 3 provides a detailed description of the problem solving process in MRPlanner, which is illustrated with a case study of a simple material transportation problem in Section 4; Section 5 concludes with some discussion.

II. PRELIMINARIES

An instance of the CSP is described by a set of variables, a set of possible values for each variable, and a set of constraints between the variables. It is formally defined as follows [5]:

Definition 1(constraint satisfaction problem) A CSP is a triple (X, D, C), where

- X is a finite set of variables $\{x_1, x_2, \ldots, x_n\}$;
- D is a function which maps each variable in X to its domain of possible values, of any type, and Dx_i is used to denote the set of objects mapped from x_i by D;
- C is a finite, possibly empty, set of constraints on an arbitrary subset of $P(X)$, or the powerset of X.

In [6] we also formally define aggregation, normal combination, and orthogonal combination of CSP, on top of which problems can be specified, proved correct, and composed together in such a way to preserve their intra-properties as well as inter-relations.

A-Team is an agent based architecture in which multiple problem solving methods (agents) cooperate to produce a set of non-dominated solutions, i.e., no feasible solution is better than a solution in the set in all the objectives. An agent in an A-Team typically consists of the following four components that enable him to participate into the teams and assist in the population evolution:

- A requester that requests notification of system events and determines if the agent has become relevant to the current situation.
- A selector that picks zero or more solutions as input to the optimization operator.
- An operator that runs and produces zero or more result solutions and passes them to the distributor.
- A distributor that filters the results and adds them to the output population.

The A-Team architecture has the capability to combine multiple solution approaches for scheduling and constraint programming. The selection of agents to run and some of the algorithms used by the agents are probabilistic. The key to getting good results is to select a divers set of algorithms and to run the A-Team until the marginal rate of improvement falls below a threshold or the available time has been exhausted. Although the results can not be guaranteed theoretically, empirical results suggest that this approach produces[7].

III. SOLUTION APPROACH

In MRPlanner, there are fives states for an MMRP solution during the problem solving process, as illustrated in Fig. 2.

A. CSP Specification Construction

The system provides a drag-and-drop user interface for the schedulers to define MMRP problems, which are composed of objectives, variables and their domains, and constraints. When a scheduler proposes an initial problem, an empty solution is created and sent to the solution space (population).

Taking an empty solution from the population, the specializer extracts the part of the initial problem whose objectives

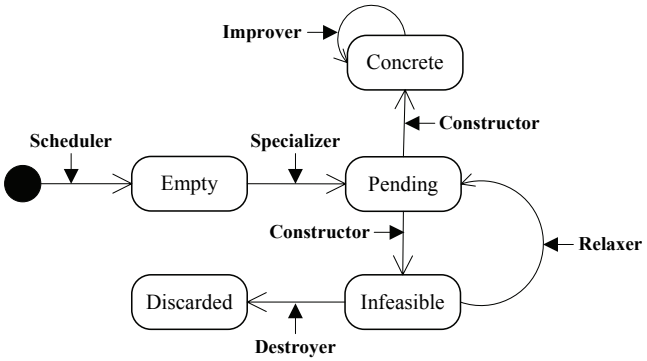

Fig. 2. The state diagram of CSP solution in MRPlanner

and constraints fall into his domain-specific category. The specializer then uses the domain-specific library to build a CSP specification for the problem part, writes the sub-specification into the solution, and sends it back to the population. Here we call a problem part together with its sub-specification as a "pending sub-solution". The specializers of other domain-specific categories continually build sub-specifications for the solution until all parts of the initial problem is built into pending sub-solutions, that is, the whole empty solution becomes a pending solution.

Currently MRPlanner implements 27 types of specializers, whose domain-specific categories can be further divided into four sections: the materials section, the resource section, the life cycle section, and the environment section, which are orthogonal to each other. As partly illustrated in Table 1, each section contains a group of MMRP templates, and each template is formally described using the structure in Definition 1. Nearly every integrated MMRP problem can be viewed as an intersection of these sections.

B. Problem Solving

A constructor takes a pending solution from the population, traverses all the pending sub-solutions and picks out those whose CSP specifications are subject to his problem-solving tactics. The constructor then tries one of his concrete algorithms to solve the sub-CSP; if successful, the pending sub-solution becomes a concrete sub-solution. Constructors with other tactics continually try their algorithms to solve pending sub-problems remaining in the solution. If all pending sub-solutions become concrete, the whole pending solution itself becomes a concrete one. After being passed by the constructors with all of the solving statics, if the solution still has any pending sub-solution, it becomes an infeasible solution.

Currently MRPlanner implements 11 types of constructors, each contains a group of concrete algorithms. As shown Fig. 3, the algorithm library is organized with a taxonomic tree [8], which contains three top-level tactics including mathematical programming (sub-tactics include integer programming, linear programming, and dynamic programming), global search (sub-tactics include the backtrack, breadth-first search, width-first search, and bound-and-branch), and local search (sub-tactics

TABLE I
DOMAIN-SPECIFIC SECTIONS AND TEMPLATES IN MRPLANNER

Section	Type	Properties
Material	Ferrous metal	iron-carbon, cast iron, alloy steel ...
	Non-ferrous metal	aluminum alloy, copper alloy, bearing alloy ...
	Macromolecular	covalent bond, molecule bond, hydrogen bond ...
	Ceramic materials	glass, ceramics, glass-ceramic...
	Composite materials	fiber reinforcement, particulate reinforcement, layered materials...
	Natural materials	rubber, graphite, asbestos, quartz ...
	Functional materials	electrical functions, magnetic functions, acoustic functions ...
	Other types	chemical pipes, metal hoses, powder metal ...
Resource	Human resource	analyst, driver, technician, repairer ...
	Materiel	construction machines, truck, cargo plane, cargo ship ...
	Equipments	measurement, machining, testing, maintenance ...
	Establishments	storehouse, workshop, dork, platform ...
	Oils	gasoline, diesel oil, engine oil ...
	Carries	packaging, load/unload, storage ...
	Computer resources	application, document, database, network ...
Life cycle	Analysis	feasibility anal., economic anal., deployment anal. ...
	Planning	main, risk, quality-assurance ...
	Implementation	design, production, acquisition, sub-contract ...
	Monitoring	quality tracing, risk control, counter plan ...
	Disposal	rebirth, abandon, demise, convey...
	Feedback	call back, recording, update...
Environment	Pollution	gaseous/liquid/solid waste...
	Energy consumed	nuclear, wind, solar...
	Resource consumed	mines, oils, industrial water...
	Rebirth	circulation, decomposition...
	Hazard	Life, animals, plants...
	Relief	catalyst, bio-preparation, absorption...

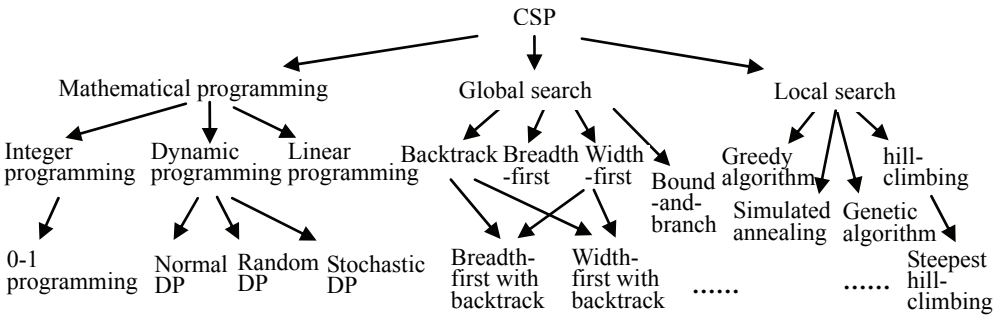

Fig. 3. The organization of the algorithm library

include the greedy algorithm, hill-climbing algorithm, simulated annealing algorithm, and genetic algorithm). In particular, global search and local search do not always guarantee the optimal solutions since their main concern is to quickly find an acceptable, sub-optimal solution for combinatorial problems in a reasonable time.

When filtering a pending sub-solution, the constructor decides whether its solving tactics fits for the problem by estimating the constraint complexity and strength and objective complexity of the CSP, as illustrated by the flowchart in Fig. 4.

The pre-defined solution library saves optimal solutions for typical CSP. When encountering a problem that is same or isomorphic to a library problem, instead of re-solving it from scratch, the constructor simply uses the library solution or builds a isomorphic solution [9]. After successfully solving a new CSP, the constructor or the improver can also add the concrete solution to the pre-defined library.

C. Solution Improving, Relaxing, and Destroying

There two types of improvers in MRPlanner: one takes single concrete sub-solution as input and try to improve it in one or more objectives; the other reallocates the resource between sub-solutions and try to improve one or more objectives for the whole concrete solution. The global constraints optimization is achieved mainly by constraint propagation [10], feasibility reasoning and optimality reasoning [11].

Each solution has two marks: $RelaxLevel$ (ranging from 0 to 2) that represents the level of allowable constraints relaxation for the problem, and $RelaxCount$ that saves the times the solution has been relaxed. Policies for solution relaxation include [4]:

- For an infeasible solution whose $RelaxLevel$ is 1 or 2, the relaxer heuristically violates some problem con-

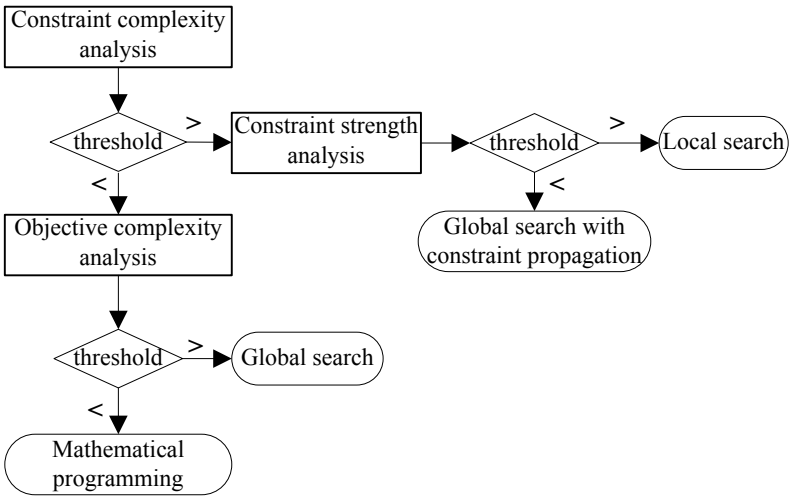

Fig. 4. The heuristic algorithm selection flow

straints based on constraint sensitivity analysis, increases its $RelaxCount$, and sends it back to the population. The solution and all its relaxed sub-solutions become pending.

- For a concrete solution whose $RelaxLevel$ is 2, the relaxer tries to violate problem constraints and tests the result: if some slight violations yield significant improvement on one or more objectives, increases its $RelaxCount$, and sends it back to the population; otherwise remains the solution unchanged.
- For a concrete solution whose $RelaxLevel$ is 1 and whose $RelaxCount$ is not zero, the improver tries to remove the constraint violations through iterative repair [12].
- For an infeasible solution whose $RelaxLevel$ is zero, or whose $RelaxCount$ exceeds a per-defined limit, the destroyer removes it from the population, that is, the solution becomes discarded.

In addition, the destroyers can detect redundant solutions where there are same solutions or better solutions for the same problem.

IV. A Simple Material Transportation Problem

Now we use a simple transportation problem to illustrate the behavior of A-Team agents in our architecture. The problem deals with transporting raw material from the sources (namely A1 and A2) to the destinations (namely E1, E2, and E3) through the network shown in Fig. 5. The edges are labeled with distances, and the source and destination vertices are labeled with quantities of supply and demand respectively.

At the first stage, the scheduler proposes the unactuated solution whose $RelaxLevel$ is 2, and then the specilizers transform it into a pending solution which contains seven sub-problems, including the chief sub-problem defined in Table 2 and six sub-problems summarized as follows:

- P_1: MIN-$f = PATH(A_1, E_1)$
- P_2: MIN-$f = PATH(A_1, E_2)$
- P_3: MIN-$f = PATH(A_1, E_3)$
- P_4: MIN-$f = PATH(A_2, E_1)$
- P_5: MIN-$f = PATH(A_2, E_2)$
- P_6: MIN-$f = PATH(A_2, E_3)$

At the second stage, some constructors with the dynamic programming algorithm and Dijkstra algorithm independently work out the concrete sub-solutions for $P_1 \sim P_6$ as follows:

- P_1: MIN-$PATH(A_1, E_1) = 12$ (A1–B1–C1–D1–E1)
- P_2: MIN-$PATH(A_1, E_2) = 11$ (A1–B1–C2–D2–E2)
- P_3: MIN-$PATH(A_1, E_3) = 28$ (A1–B1–C2–D2–E3)
- P_4: MIN-$PATH(A_2, E_1) = 13$ (A2–B1–C1–D1–E1)
- P_5: MIN-$PATH(A_2, E_2) = 11$ (A2–B3–C3–D2–E2)
- P_6: MIN-$PATH(A_2, E_3) = 28$ (A2–B3–C3–D2–E3)

Now all the orthogonal sub-problems of P_0 are solved, and then a constructor with the simplex algorithm works out the concrete sub-solution for P_0 (as shown in Table 3), and the whole solution becomes a concrete one.

At the third stage, since the $RelaxLevel$ of the solution is 2, a relaxer performs simplex sensitivity analysis on P_0, relaxes its constraint according to RELAXATION defined in Table 2, and works out a solution that yields a saving of 340, as shown in Table 4.

V. Conclusion

Constraint programming is capable of handling large and combinatorial CSP, and therefore has been successfully applied in areas of planning and scheduling. In combination with agent techniques, its capability to represent complex constraints and its intelligence in problem solving can be improved tremendously.

The paper reports MRPlanner, an agent-based constraint programming system for solving large, particularly combinatorial, problems in MMRP. It uses an A-Team architecture in which multiple problem solving agents cooperate with each other by exchanging results to produce a set of non-dominated solutions. MRPlanner features separation of concerns in specifications, domain-specific optimization, and

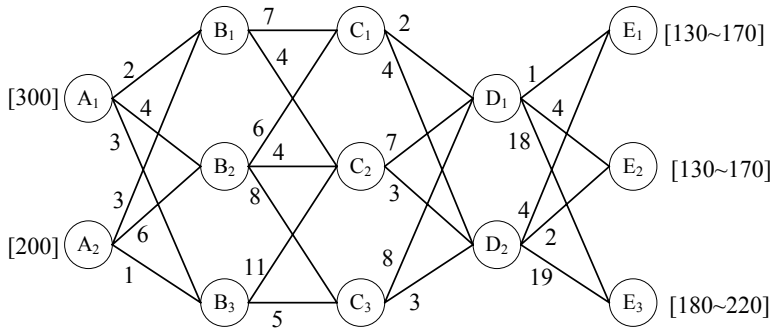

Fig. 5. The transportation network

TABLE II
SUB-PROBLEM P_0

PROBLEM	P_0
ORTHOGONALIZING	P_1: $c_{11}\#P_1.OBJ(0)$
	P_2: $c_{12}\#P_2.OBJ(0)$
	P_3: $c_{13}\#P_3.OBJ(0)$
	P_4: $c_{21}\#P_4.OBJ(0)$
	P_5: $c_{22}\#P_5.OBJ(0)$
	P_6: $c_{23}\#P_6.OBJ(0)$
OBJECTIVES	MIN: $f = \sum_{i=1}^{2}\sum_{j=1}^{3} c_{ij}x_{ij}$
CONSTRAINTS	C1: $x_{11} + x_{12} + x_{13} = 300$
	C2: $x_{21} + x_{22} + x_{23} = 200$
	C3: $x_{11} + x_{21} = 150$
	C4: $x_{12} + x_{22} = 150$
	C5: $x_{13} + x_{23} = 200$
	C6: $\forall(0 < i \leq 2, 0 < j \leq 3)x_{ij} \geq 0$
RELAXATIONS	C3: ± 20, C4: ± 20, C5: ± 20

TABLE III
THE CONCRETE SOLUTION FOR P_0

	E1	E2	E3	Supply
A1	150	0	150	300
A2	0	150	50	200
Demand	150	150	200	500
Result	9050			

TABLE IV
THE REVISED CONCRETE SOLUTION FOR P_0

	E1	E2	E3	Supply
A1	150	150	0	300
A2	0	20	180	200
Demand	150	170	180	500
Result	8710			

automatic generation of high-performance and reliable solutions. The system has been successfully applied in real-world problem solving and has demonstrated its contribution to the significant improvement of the MMRP efficiency and effectiveness. Ongoing efforts including adding a workshop domain-specific section to enable materials scheduling within the process of manufacturing, and investigating more adaptive behaviors of the system agents.

REFERENCES

[1] S. Bistarelli, "Semirings for Soft Constraint Solving and Programming," *Lecture Notes in Computer Sciences* , vol. 2962, pp. 1-20, 2004.

[2] K. M. Alguire, and C. P. Gomes, "Technology for Planning and Scheduling under Complex Constraints," in *Proceedings of Society for Optical Engineering*, Boston, USA 1997.

[3] S. N. Talukdar, P. Souza, and S. Murthy, "Organization for computer-based agents," *Engineering Intelligent Systems* , vol. 1, no. 2, pp. 75-87, 1993.

[4] Y. J. Zheng, L. L. Wang, and J. Y. Xue, "An A-Team Based Architecture for Constraint Programming," *Lecture Notes in Computer Science* , vol. 4088, pp. 552-557, 2006.

[5] E. P. K. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, London, UK, 1993.

[6] Y. J. Zheng, L. L.Wang, and J. Y. Xue, "A Constraint Programming Framework for Integrated Materiel Logistic Support," *Journal of Nanjing University* , vol. 40, no. s10, pp. 30-35, 2005.

[7] R. Akkiraju, P. Keskinocak, S. Murthy, and W. Frederick, "An Agent-Based Approach for Scheduling Multiple Machines," *Applied Intelligence* , vol. 14, no. 2, pp. 135-144, 2001.

[8] D. R. Smith, "oward a Classification Approach to Design," *Lecture Notes in Computer Sciences*, vol. 1101, pp. 62-84, 1996.

[9] Y. J. Zheng and J. Y. Xue, "MISCE: A Semi-Automatic Development Environment for Logistic Information Systems," in *Proceedings of 1st IEEE International Conference on Service Operations and Logistics, and Informatics*, Beijing, China, pp. 1020-1025, 2005.

[10] P. Pepper and D. R. Smith, "A High-level Derivation of Global Search Algorithms (with Constraint Propagation)," *Science of Computer Programming*, vol. 28, special issue on FMTA (Formal Methods: Theory and Applications) pp. 247-271, 1996.

[11] F. Focacci, A. Lodi, and M. Milano, "Optimization-Oriented Global Constraints," *Constraints* , vol. 7, no. 3-4, pp. 351-365, 2002.

[12] M. Zweben, B. Daun, E. Davis, and M. Deale, "Scheduling and rescheduling with iterative repair," *Intelligent Scheduling* (eds. Zweben, M. and Fox, M.S.), Morgan Kaufman, pp. 241-255, 1994.