# Computer-Aided Reconfiguration Planning: An Artificial Intelligence-Based Approach

**Li Tang**

**Yoram Koren**

University of Michigan,
Ann Arbor, MI 48109

**Derek M. Yip-Hoi**
University of British Columbia,
Vancouver, B.C., V6T 1Z4, Canada

**Wencai Wang**
University of Michigan,
Ann Arbor, MI 48109

*The manufacturing industry today faces a highly volatile market in which manufacturing systems must be capable of responding rapidly to market changes while fully exploiting existing resources. Reconfigurable manufacturing systems (RMS) are designed for this purpose and are gradually being deployed by many mid-to-large volume manufacturers. The advent of RMS has given rise to a challenging problem, namely, how to economically and efficiently reconfigure a manufacturing system and the reconfigurable hardware within it so that the system can meet new requirements. This paper presents a solution to this problem that models the reconfigurability of a RMS as a network of potential activities and configurations to which a shortest path graph-searching strategy is applied. Two approaches using the A* algorithm and a genetic algorithm are employed to perform this search for the reconfiguration plan and reconfigured system that best satisfies the new performance goals. This search engine is implemented within an AI-based computer-aided reconfiguration planning (CARP) framework, which is designed to assist manufacturing engineers in making reconfiguration planning decisions. Two planning problems serve as examples to prove the effectiveness of the CARP framework.*
[DOI: 10.1115/1.2218369]

## 1 Introduction

Manufacturers are currently facing a market that is characterized by a short window of opportunity for new products and large fluctuations in product demand. The introduction of computer aided design (CAD) and concurrent engineering technologies has drastically reduced product development time. However, a manufacturing system that can quickly respond to product change and produce new products with short lead time is critical to the manufacturing industry in a global competitive environment.

One option is to apply reconfigurable manufacturing systems (RMS) that can be upgraded through future reconfigurations to increase capacity and change functionality [1]. Since manufacturers are being forced to handle this type of change-over more frequently, it is critical for them to plan the reconfiguration activities on the system and its components in a cost-effective way. For example, given a RMS such as the one in Fig. 1 reconfiguration planning will determine if machines need to be added or removed, if the system layout and flow paths need to be changed and how tasks should be reallocated to best produce a new part mix at the cheapest cost. Similarly for a reconfigurable machine tool (RMT) such as the one in Fig. 2 reconfiguration planning may result in new spindles being added or removed and existing ones repositioned to account for new features that must be machined.

The purpose of this paper is to present progress on the creation of a computer-aided reconfiguration planning (CARP) framework that will assist manufacturing engineers in making reconfiguration planning decisions. The paper is organized as follows. Section 2 reviews relevant research. Section 3 defines terminology and establishes a generic model for a reconfigurable object. An AI-based CARP framework for reconfigurable manufacturing is then developed in Sec. 4. Two examples are examined in Sec. 5 to prove the effectiveness of our methodology. Some future research directions are described in Sec. 6 and Sec. 7 concludes the paper.

Contributed by the Engineering Simulation and Visualization Committee of ASME for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received November 23, 2004; revised manuscript received February 20, 2006. Assoc. Editor: R. Crawford.

## 2 Relevant Research

The configuration of a manufacturing system is a very important strategic decision that many companies have to consider in their business cycles. There exist two distinct areas in the study of manufacturing systems: configuration design for a new system and operation planning for an existing one during its operational period. The former problem has been investigated by numerous researchers who model it as an optimization problem that seeks the best configuration and the respective task allocation. Askin and Zhou [2] proposed a method to find optimal machine requirements and the task sequence for flow-line cells by solving the shortest path problem. Kimms [3] formulated a NP-hard problem of finding a flow-line configuration such that the net present value of cash outflows for installing and maintaining the flow line is minimized. Donohue and Hopp [4] developed a line design algorithm using a dynamic programming approach to choose the number and type of machines for each stage such that the cost is minimized. Tang and Yip-Hoi [5] presented a genetic-algorithm-based approach to select concurrently machines and buffers as well as identify task allocations for a multipart manufacturing system.

As for the operational planning problem, most researchers place emphasis on system control issues. Few consider the issue of how an existing system can be reengineered or reconfigured in order to accommodate market changes. Bradford and Childe [6] propose a nonlinear redesign methodology for manufacturing systems which constructs an iterative strategic model to assist system redesigns for continuous improvement. Aiken and Hodgson [7] link system reengineering (SR) efforts to business process reengineering (BPR) and they present an integrated BPR and SR model which offers some guidance for the users to redesign a system and leverage reengineering investments. Chan and Juang [8] implement a BPR approach for manufacturing enterprises by applying FMS design and analysis technologies to cope with technical and business changes. Note that this research approaches the problem on the strategic or conceptual level. For a RMS designed with the capability of adjusting product mixes and producing new products, there is a need for computer-aided methodologies to plan reconfiguration activities for the users.

  

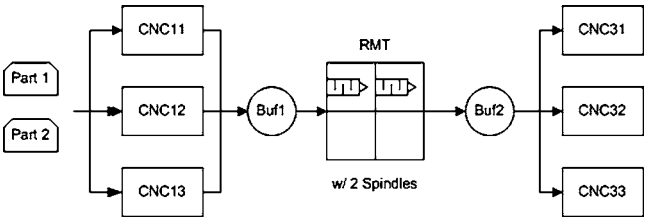


**Fig. 1 The configuration of a RMS**

The reconfiguration of a RMS involves changes to both the hardware components of the system such as RMTs and fixtures, and the system configuration itself. Researchers have presented specific solutions for both reconfigurable devices and systems. Moon and Kota [9] applied the screw theory to automatically build a RMT with appropriate modules according to functionality requirements. Son [10] presented an approach to generate the RMS configuration for a single product from alternative machining system families. However, a generic method that can transcend both device and system level reconfigurations is absent from the literature.

As it will be explained later in this paper, the reconfiguration planning problem can be modeled as a shortest path graph-searching problem in which an optimal path consisting of a series of reconfiguration activities is to be found in order to achieve an expected goal. Two groups of algorithms are used to solve the shortest path problem. One is so called as a "blind search" where the algorithm does not rely on any information about the cost of the path to the goal in selecting the next node to expand. Such algorithms are breadth-first, depth-first, and Dijkstra's algorithm [11]. The other group is "informed-search" algorithms such as the best-fit and $A^*$ algorithm, which expand the node based on a heuristic estimate of the cost to the goal. Among these search strategies, the $A^*$ algorithm has some advantages over the others considering both computing time and memory space efficiency [12]. Cherif and Gupta [13] employ the $A^*$ algorithm to develop a motion planner for manipulation of a multifinger robotic hand. In addition, Nilsson [14] proved that the $A^*$ algorithm will guarantee an optimal solution if the heuristic function adopted by the algorithm satisfies the admissibility property. Due to these advantages, it is utilized in this research within the plan-searching procedure.

## 3 Planning for Reconfiguration

**3.1 Reconfiguration Actions and Processes.** A *reconfiguration action*, $a_i$ is an operation that changes the structure of a reconfigurable object so enabling it to vary its performance (functionality, capacity, etc.) from that of its previous state. These actions can change both the relationship between and the internal structure of the components that make up the object. Such components are themselves reconfigurable. Hence, a reconfiguration action behaves recursively operating on many levels of components before completing its work on the object to which it is initially applied.

A *reconfiguration process* $G=\langle a_1, a_2, \ldots, a_p \rangle$ is a series of reconfiguration actions that transform a reconfigurable object from one state to another state to realize a change in its performance. Note that the specific order of actions in addition to being a linear sequence can also contain parallel actions.

**3.2 Reconfiguration Planning.** *Reconfiguration planning* is the activity of systematically identifying the best reconfiguration process for changing a reconfigurable object from an initial state to a final state in order to satisfy a new performance goal.

The reconfiguration planning problem is a four-tuple $(O, S_I, Y_F, G)$ where $O$ is the reconfigurable object, $S_I$ is the initial state of $O$, $Y_F$ is the expected performance goal, and $G$ is a reconfiguration process to be identified such that $Y_F$ can be realized by applying the actions in $G$.

**3.3 Reconfigurable Objects.** Reconfiguring a RMS can impact every level of the system. To develop a planning framework applicable to all levels, it is necessary to extract common properties and define a generic model for all levels of objects in the RMS. For this purpose, an artificial intelligence (AI)-based approach is applied in order to extend to different domains.

An *object* is an identifiable item such as a device, a program, or a system that has its own unique structure, behavior, and capabilities that satisfy specific performance requirements. Objects can be recursively contained, meaning that an object can own other objects referred to as member objects. For example, a RMS contains reconfigurable components such as machines, fixtures, or cutting tools.

A *class* is a set of objects with similar structure and behavior. For example, RMTs with similar structure and functionality can be collected and put into a RMT class.

A *reconfigurable object* is an object whose structure and state can be modified by a set of actions to realize changes in its performance. The behavior of a reconfigurable object is affected by the external environment and the limitations of its internal components. A reconfigurable object consists of the following elements:

- *Member Object*, $o_i$:
  A component of a reconfigurable object i.e., $o_i \in O$, $i = 1, 2, \ldots, m$. The member object typically though not always belongs to a different class from its parent object.
- *State*, $S_i$:
  A state describes the current condition of an object, including relationships between its member objects and their conditions. It is defined as

$$S_i = [p(\nu, o_i, \langle o_j \rangle), \nu]$$

  Here, $p(\nu, o_i, \langle o_j \rangle)$ is a set of propositions each either true or false based on the values of the state variables $\nu \subseteq \mathbf{R^m}$. They define the condition of a member $o_i$ object or optionally its relationship with another member object $o_j$. More than one proposition can be defined for each object or relationship.
- *Constraint*, $C_i$:
  A constraint defines the domain of a state variable using mathematical formulae like an inequality $g(\nu) \geq 0$, or alternately it may be applied to override the value returned by a proposition to modulate its effect. For example, constraining a proposition to be always true can have the result of fixing the configuration of an object. Conversely constraining a proposition to be always false can be used to define an infeasible configuration. A constraint set $C_i$ is defined as

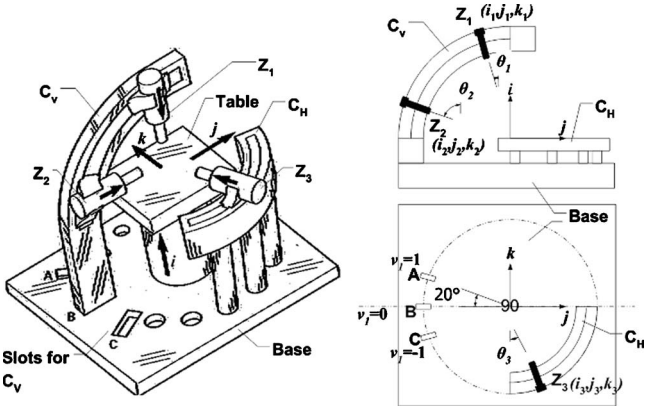


**Fig. 2 A multispindle RMT**

$$C_i = \{q = \text{True} \| \text{False} | q \in p(\nu, o_i, \langle o_j \rangle)$$
$$\cap \{g(\nu) \geq 0 | \nu \in \mathbf{R^m}\}$$

- *Performance Metrics*, $Y$:
  Performance metrics are a vector of values each providing a measure for some functionality that an object possesses. The performance metrics are represented by $Y=(y_1, y_2, y_3, \ldots)$, $y_i \in \mathbf{R^m}$.
- *Set of Reconfigurable Actions*, $A$:
  Reconfigurable actions that make up a reconfiguration plan are selected from the superset of actions defined over all reconfigurable objects. Each action can be performed only if a set of preconditions are satisfied. After an action is completed state variables are changed and propositions are re-evaluated as either true or false. The resources required for the action are also specified using performance metrics such as reconfiguration cost or time. A set of actions for a reconfigurable object $O$ is defined as
  $$A = \{a_1, a_2, \ldots, a_n\}.$$
  Each action $a_i$ is defined by a six-tuple
  $$a_i = (\arg_i, \text{pre}_i, \text{add}_i, \text{del}_i, \nu_i, c_i)$$
  where $\arg_i$: arguments of the action $a_i$ specifying the numerical state variables that will be modified by the action; $\text{pre}_i$: preconditions of the action $a_i$, represented by a set of propositions; $\text{add}_i$: a set of propositions that become true after executing $a_i$; $\text{del}_i$: a set of propositions that become false after executing $a_i$; $\nu_i$: the state variable changed by $a_i$, $\nu_i \subseteq \nu(S)$; and $c_i$: resource consumed by action $a_i$, like cost or time.
- *Mapping Function*
  A mapping function $\Phi$ is a one-to-one or many-to-one correspondence relationship between the states and the performance of the object, $Y=\Phi(S)$.
- *Rules*

Some heuristic knowledge and expertise can be applied to assist the derivation of a reconfiguration plan for the specified performance goals. The knowledge is represented as a set of rules linked to the object. According to their usage, the rules are classified into three categories: validity rules, arbitration rules, and control rules. *Validity rules* are used to check if the expected performance goals have exceeded the capability of the reconfigurable object. *Arbitration rules* determine actions that are exactly needed to achieve the performance goal by reconfiguring the object from its current state. If arbitration rules cannot be easily defined for the problem, a set of *control rules* can be applied to guide the planner to search for a feasible path from the initial state to the final goal. The control rules cannot guarantee a successful plan but they will expedite the searching process.

A rule is a statement of the form: **If** ⟨**x**⟩ **Then** ⟨**y**⟩ **Else** ⟨**z**⟩, where **If** part is the rule premise or condition, and the **Then** part is the consequence or action. The **Else** component of the consequence is optional. The rule fires when the **If** part is determined to be true.

Figure 3 shows an example of the reconfigurable object, which is a three-fingered reconfigurable fixturing system [15]. The fixturing system consists of a fixed *Module1* and a movable *Module2*. *Module1* has two fingers positioned along the circumferences of two adjacent circles, while *Module2* has a single finger that can be adjusted along a slot. A planar workpiece can be immobilized by repositioning three fingers.

The reconfigurable object is represented by

$$\text{RFS} = \{Module1, Module2, Finger1, Figure2,$$
$$Figure3, Workpiece\}.$$

Its state can be represented by the locations of three fingers

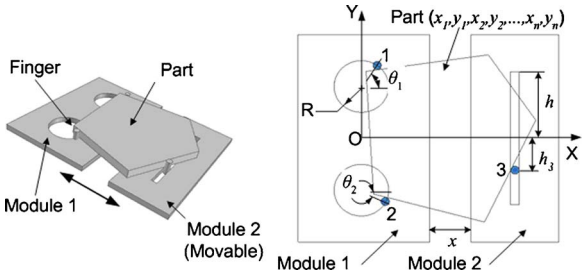


**Fig. 3 A three-fingered RFS**

$(\theta_1, \theta_2, h_3)$, the distance $x$ between *Module1* and *Module2* and the coordinates of the polygon vertices of the planar workpiece $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

$$S = \{p1 = (Finger1, Module1, \theta_1), p2 = (Finger2, Module1, \theta_2),$$
$$p3 = (Finger3, Module2, h_3), p4 = (Module1, Module2, x);$$
$$\theta_1, \theta_2, h_3, x, \{x_1, y_1, x_2, y_2, \ldots x_n, y_n\}\}$$

The constraints are: {all propositions $p1, p2, p3$, and $p4$ must be true} $\cap \{0 \leq \theta_1 \leq \pi, 0 \leq \theta_2 \leq \pi, -h \leq h_3 \leq h\}$.

The actions include: $a_1$: repositioning *Finger1* in the angle $\theta_1$; $a_2$: repositioning *Finger2* in the angle $\theta_2$; $a_3$: moving *Finger3* to the height $h_3$; and $a_4$: moving *Module2* to the position $x$.

The performance $Y$ is a logical value that indicates if the planar object is immobilized on the fixture. Thus the mapping function $\Phi$ returns a TRUE or FALSE value according to the state variables $\theta_1$, $\theta_2$, $h_3$, $x$ and $\{x_1, y_1, x_2, y_2, \ldots, x_n \ldots y_n\}$.

$$Y = \Phi(\theta_1, \theta_2, h_3, x, \{x_1, y_1, x_2, y_2, \ldots x_n, y_n\})$$

**3.4 CARP.** Reconfiguration planning is a complicated problem, which like process planning takes significant effort and time for a human reconfiguration planner to perform. The work largely depends on human intelligence and experience. In addition, the complexity makes it prone to error and variability. Without standardization of best practices each planner can generate a different solution based upon their own understanding and experience.

CARP is presented in this paper as a mechanism to assist the human reconfiguration planner and to aid in the collection and standardization of best practices for reconfiguration. CARP provides an automated methodology for creating a reconfiguration plan for a reconfigurable object using computerized techniques. In this research the object being reconfigured can either be a reconfigurable mechanical device like the RMT mentioned previously, or a complex system like a RMS which consists of many reconfigurable objects. However, it need not be limited to this.

The basic input to a CARP system includes the description of the reconfigurable object to be changed, its current state, and the new performance goals to be achieved. The description of the object including feasible reconfiguration actions is needed by the planner to efficiently and correctly plan the reconfiguration of the object. Also needed is knowledge in the form of rules for how to select appropriate actions, evaluate propositions, and apply constraints during the planning. From these inputs, the CARP system will output a series of reconfiguration actions with a specific order to meet the desired change in RMS performance.

In the following section an overview of a framework for a CARP system will be presented.

## 4 CARP Framework

A CARP framework has been developed. It consists of two primary modules for model-building and plan-generation as shown in Fig. 4. Each comprises a number of tasks that will be described in the following sections.

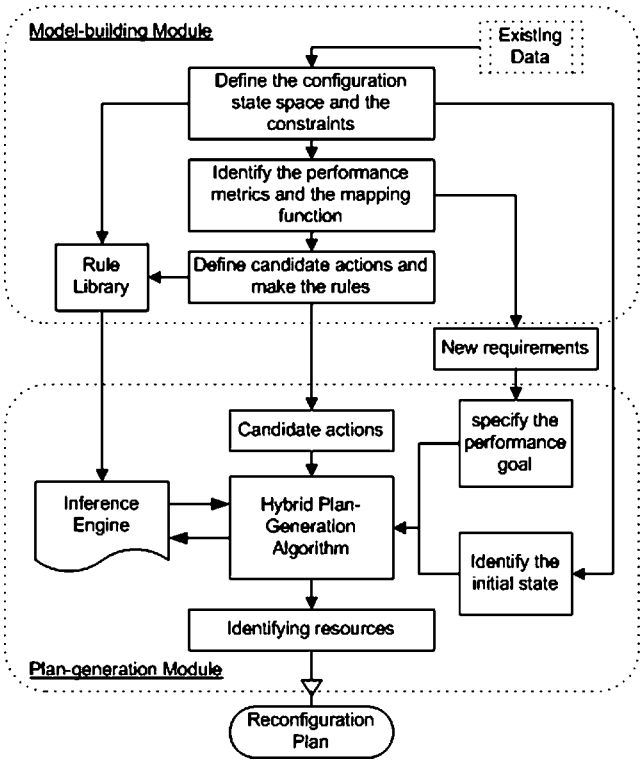


**Fig. 4 AI-based CARP framework**

**4.1 Model-Building Module.** The main purpose of the model-building module is to collect information and construct instances of reconfigurable objects using the model described in the last section. The information of interest includes four parts: (i) the configuration of the object, (ii) the performance metrics of the object, (iii) the actions to reconfigure the object, and (iv) the knowledge (rules and constraints) about the reconfigurability of the object. This module is executed once unless new information is provided for an object. After execution, a model of the reconfigurable object is created and the relevant knowledge is stored in a knowledge-base (rule library) that is maintained in the framework throughout future planning stages.

The first task is to define a state space that describes all possible configurations of the object. Each state represents a configuration of the object (the components or member objects belonging to the base object, the propositions and numerical variables defining the relations among the components). The constraints, which are expressed as mathematical formulae of numerical state variables are also specified in this step and stored in the knowledge base. This activity is accomplished through extracting domain-dependent knowledge relevant to the reconfigurability of the object from existing engineering or organizational data.

The ensuing task is to specify the performance metrics and identify the mapping function between the state and the performance. For problems in the reconfigurable manufacturing domain, one usually defines productivity and/or functionality as the performance metrics. For example, the performance criteria of a RMT may include the number of cutting-tool-access-directions and the cycle time for a set of tasks. The mapping function is determined by the human planners according to their knowledge and expertise. It can be either an analytical function or a simulation-based procedure which computes the performance for a given state of the object.

Human planners need to identify candidate actions that can change an object's state. They can eliminate actions based on their experience and judgment. For example, if an action requires too many resources a planner can remove it from consideration as a candidate. To define a candidate action, the effect of executing it and the preconditions required to trigger its execution must be specified. The required resources will also be determined. All the information related to an action is stored in the knowledge base. These rules are used by the plan-generation module to guide the derivation of a reconfiguration plan.

**4.2 Plan-Generation Module.** The plan-searching module identifies the necessary actions with proper sequence that can achieve the expected performance goal. The generation process is based upon the information collected with the model-building module.

The first task is to identify the initial object state and specify an expected performance goal as inputs. Then a hybrid plan-generation algorithm will select a sequence of actions from the candidate actions to achieve the goal. The term "hybrid" means that the planner chooses between using arbitration rules and an $A^*$ plan-searching algorithm constrained by control rules to generate a plan. A hybrid methodology is proposed because the solution to some planning problems can always be captured by rules defined from experience. Other problems have many feasible solutions that can only be differentiated by applying an optimization strategy that searches for the best. A hybrid planner provides the greatest flexibility in solving a range of problems.

In both cases the planner interacts with the knowledge base through an inference engine, which reads the rules that are stored and fires the applicable ones to make decisions as the algorithm proceeds. The firing of rules will perform checks to see if a goal is achievable (validity rules), result in plan generation using purely arbitration rules, and decide if actions can be performed or a configuration/state is feasible through the application of control rules during a search for the best plan.

**4.3 Extracting Reconfigurability From Existing Data.** Currently, the process of building a reconfigurable object model is accomplished by planners using their expertise and human intelligence. It is a manual activity without any automated tools to assist decision making.

However, it is possible that data from existing digital sources can be transformed into a reconfigurable object model either automatically or semi-automatically if these sources are augmented with additional information. For reconfigurable mechanical devices like a RMT or a reconfigurable fixture, their CAD assembly models maintain the basic structure to which information can be added that is useful in building the reconfigurable object model. For a RMS, a virtual factory model consisting of 3D digital mockups of plant facilities can be used in the same way to capture reconfigurability. Further research is needed to identify efficient methods for constructing and managing additional information to support digital models of reconfigurable objects. This problem is not addressed in this paper.

**4.4 A Hybrid Algorithm for Plan Generation.** In this section, a hybrid plan-generation algorithm that applies two methods for creating a reconfiguration plan is introduced. It either applies arbitration rules when applicable, to generate a reconfiguration plan that satisfies the performance goal, or it employs an $A^*$ plan-searching algorithm to search for an optimal plan.

*4.4.1 Flowchart of the Algorithm.* The hybrid plan-generation algorithm is shown in Fig. 5.

First, the initial state of the object is identified and the performance goal to be achieved is specified. Then the validity rules evaluate the performance goal and decide whether it is accomplishable or not. If any one of the rules is fired, the goal will be rejected and an empty plan is returned. As long as the goal passes the validity check, the next step is to examine whether or not the reconfigurable object preserves the arbitration rules from which a plan can be directly developed without performing a search procedure. The arbitration rules may come from either mathematical relationships that exist between the state and the performance or

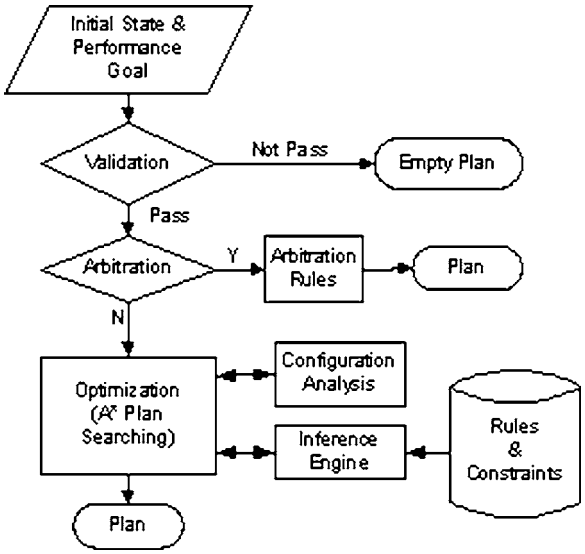


**Fig. 5 A hybrid plan-generation algorithm**

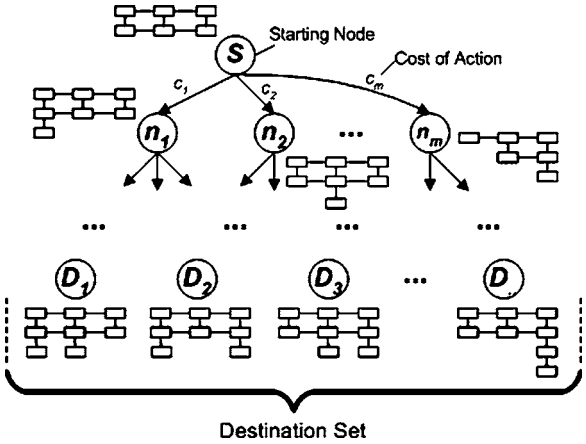


**Fig. 6 Analogy to a shortest path graph-search problem**

from the experts' experience and knowledge. If these arbitration rules exist, a plan can be derived from them directly. Otherwise, an A* plan-searching algorithm will be applied. The algorithm interacts with the control rules to determine the appropriate actions to be performed for the current state. Meanwhile, it has to evaluate the constraints to ensure the correctness of the resulting plan.

*4.4.2 Validity and Arbitration Rules*. As the initial state of the reconfigurable object and the expected performance goal are specified, a set of validity rules are used to check if the goal exceeds the capability of the object. A performance goal will be rejected if any validity rule is fired. As a result, the algorithm returns an empty plan.

After the goal passes the validity checking process, it enters the arbitration process which directly generates a reconfiguration plan if arbitration rules can be applied to the input conditions. Arbitration rules determine the actions that are needed to reconfigure an object to a state that meets the new performance goal. Normally, the arbitration process is accomplished using one of two techniques: there exists a mathematical function that defines a one-to-one relational mapping from a state to a performance goal; or the human planner has enough knowledge and previous experience to manually create a plan. Consequently, there are two ways to create arbitration rules. The first is to build rules that identify the actions needed to map from the initial state to the expected state based upon one-to-one mapping functions. For the second approach, the human planners represent their expertise or experience in the format of rules that determine what actions need to be performed to achieve the goal. If in the end no arbitration rules can be found to generate a valid plan, a set of control rules are applied by an A* algorithm to search for a reconfiguration plan.

*4.4.3 A* Plan-Searching Algorithm*. A reconfiguration planning problem can be formulated as a shortest-path graph-search problem, which is illustrated using a RMS example as shown in Fig. 6.

Each system configuration, i.e., a state of the reconfigurable object, is a node of a graph. Each arc projected from the node represents a reconfiguration action that can be performed on that state. The node pointed to by the arc represents the state after executing the action. The initial system configuration is the starting node of the graph. A set of system configurations fulfilling the performance goal make up the destination set the search will end with. A path from the starting node to one of the destination nodes represents a reconfiguration plan. The objective of the A* algorithm is to find a path with a minimal cost.

The A* algorithm can guarantee the optimality as long as the admissibility condition is satisfied, that is, a heuristic function $h'(n)$ used in the search never overestimates the actual minimal cost $h(n)$ from node $n$ to the destination (for more details refer to Nilsson [14]).

The actual optimal cost is defined as

$$f(n) = g(n) + h(n)$$

where $g(n)$=the actual reconfiguration cost to change the initial state to state $n$ (node $n$); $h(n)$=the actual minimal cost from node $n$ to one of the destination nodes. Since this value is difficult to determine, a heuristic function, $h'(n)$, is used to estimate it, and to approximate the actual optimal cost $f(n)$. It is defined as

$$h'(n) = \rho^*[\delta(n)/\delta(0)]$$

where $\rho$=the lower bound of total reconfiguration cost, which is approximated by ignoring such issues as machine reliability and system imbalance; $\delta(0)$=the difference between the initial performance and the expected goal, which is measured using the unit of the performance metrics; $\delta(n)$=the difference between the current performance at node $n$ and the expected goal, which is measured using the unit of the performance metrics.

An estimate of the actual optimal cost $f'(n)$ is

$$f'(n) = g(n) + h'(n) = g(n) + \rho\frac{\delta(n)}{\delta(0)}$$

Note that $\delta(0)$ is a constant and $\delta(n)$ only depends on $n$. As long as a small enough $\rho$ is selected such that $h'(n) \leq h(n)$, which means that $h'(n)$ is admissible, the A* algorithm will guarantee an optimal solution.

The following describes a heuristic to determine a feasible $\rho$ value.

Assume there is a set of available actions $a_1$, $a_2$,..., $a_m$, with costs $c_1$, $c_2$,..., $c_m$, respectively (refer to Sec. 3.3). Define $d_i$ as the upper bound of the performance gain if applying action $a_i$. Normally $d_i$ is calculated by applying $a_i$ on the initial system configuration.

Let $r = \max_{1 \leq i \leq m}\{d_i/c_i\}$, which stands for the maximal performance gain per unit of reconfiguration cost. Since $\delta(n)$ is the performance gap at node $n$, the actual cost $h(n)$ to the destinations will not be less than $\delta(n)/r$. That is

$$\delta(n)/r \leq h(n)$$

Define $\rho = \delta(0)/r$. It can be proven that $h'(n) \leq h(n)$.

*Proof.*

$$h'(n) = \rho^*[\delta(\mathbf{n})/\delta(0)] = [\delta(0)/r] * [\delta(n)/\delta(0)] = \delta(n)/r \leq h(n)$$

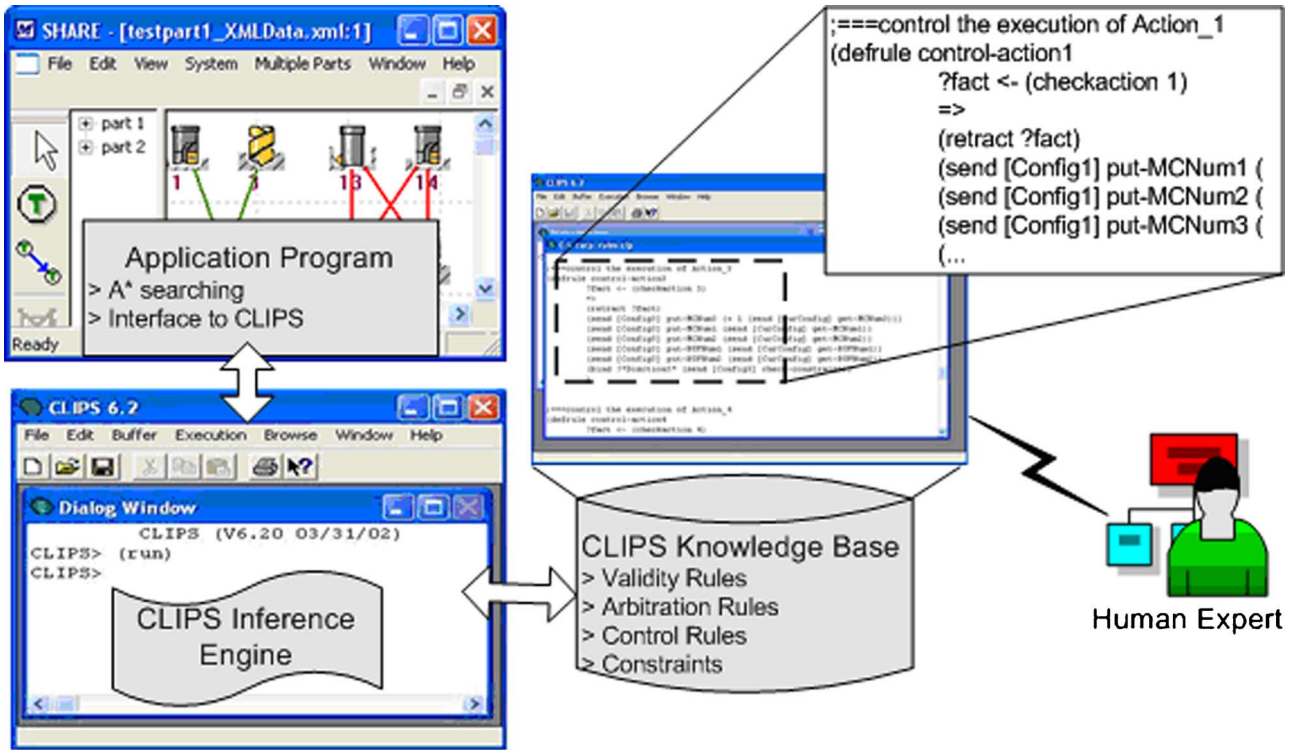


**Fig. 7 Implementation of a CARP prototype application**

It is possible to apply other simple heuristics to determine the lower $\boldsymbol{\rho}$ value. As long as the admissibility condition holds, the A* algorithm guarantees optimality.

**4.5 Implementation.** A prototype application has been developed that integrates a rule-based knowledge base, an inference engine, and an application program corresponding to the specific reconfiguration domain, as shown in Fig. 7.

The application program includes an implementation of the A* algorithm and an interface with the rule inference engine. It is written in the C++ language based on object oriented programming architecture, which makes it easier to migrate to other domains.

The knowledge base and the inference engine are implemented using CLIPS, an open source toolkit for constructing rule based expert systems [16]. The inference engine is a generic procedure implemented within CLIPS while the knowledge base is specific to the application domain. The human expert can create/edit/save the rules and constraints in the knowledge base. If the domain changes, a new knowledge base needs to be constructed.

## 5 Examples

Two examples have been examined to test the CARP framework. One example adopts arbitration rules to derive a reconfiguration plan directly while the other uses control rules and the A* plan searching algorithm to find an optimal plan.

**5.1 RMT Reconfiguration Planning.** The first example is a multispindle RMT shown in Fig. 2. In this machine three cutting tools are attached to spindles ($Z_1$, $Z_2$, $Z_3$). The part on the *Table* can move in two directions $X$ and $Y$. Spindles that can move along the arc slots of two columns $C_H$ and $C_V$. The column $C_v$ can be put in either slot $A$, $B$, or $C$.

The reconfigurable object model is defined as follows:

$$O_{\mathrm{RMT}} = \{Base, Table, Z_1, Z_2, Z_3, C_H, C_V\}$$

The state of the RMT is

$$S = \{p1 = (C_V Z_1 \theta_1), p2 = (C_V Z_2 \theta_2), p3 = (C_H Z_3 \theta_3), p4 = (C_V\ Base\ v_1), \theta_1, \theta_2, \theta_3, v_1\}$$

The constraints are

$$C = \{q1 = (C_V\ Z_1\ \theta_1)\text{:Spindle } Z_1 \text{ only can be on the vertical column } C_V\} \cap \{q2 = (C_V\ Z_2\ \theta_2)\text{:Spindle } Z_2 \text{ can only be on the vertical column } C_V\} \cap \{q3 = (C_H\ Z_3\ \theta_3)\text{:Spindle } Z_3 \text{ can be on the horizonatal column } C_H\} \cap \{q4 = (Base\ C_V\ v_1)\text{:Vertical column } C_V \text{ must be on the Base}\} \cap \left\{0 \leqslant \theta_1 \leqslant \frac{\pi}{2}, 0 \leqslant \theta_2 \leqslant \frac{\pi}{2}, 0 \leqslant \theta_3 \leqslant \frac{\pi}{2}, \theta_1 \neq \theta_2\right\} \cap \{-1 \leqslant v_1 \leqslant 1, v_1 \in Z\}$$

Here $\theta_1$ and $\theta_2$ are the angles between $Z_1$, $Z_2$, and $i$ axis, respectively. $\theta_3$ is the angle between $Z_3$ and $-k$ axis. The values of $v_1$, 1, 0, and −1, correspond to the slot $A$, $B$, and $C$ respectively.

The candidate actions are: $a_1$: move the spindle $Z_1$ along $C_V$ to a new angle $\theta$; $a_2$: move the spindle $Z_2$ along $C_V$ to a new angle $\theta$; $a_3$: move the spindle $Z_3$ along $C_H$ to a new angle $\theta$; and $a_4$: relocate the column $C_V$ to the new slot $v$.

The cutting directions of the RMT are its performance metric, expressed as

$$Y = (i_1, j_1, k_1, i_2, j_2, k_2, i_3, j_3, k_3)$$

$(i_1, j_1, k_1)$, $(i_2, j_2, k_2)$ and $(i_3, j_3, k_3)$ are three cutting directions in the $i$-$j$-$k$ coordinate system shown in Fig. 2.

Since in this example the RMT is a mechanical assembly which possesses a direct mapping relationship between the state and the

**Table 1 Rule library of the RMS**

| Rules | IF | THEN | Description |
|---|---|---|---|
| Validity rule 1 | $i_3 \neq 0$ | Goal cannot be achieved | $Z_3$ must-be-parallel to the table |
| Validity rule 2 | $\arctan(-k_1/j_1) \neq \arctan(-k_2/j_2)$ | Goal cannot be achieved | $Z_1$ and $Z_2$ must mount to $C_V$ |
| Validity rule 3 | $\arccos(i_1/\sqrt{i_1^2+j_1^2+k_1^2}) \equiv \arccos(i_2/\sqrt{i_2^2+j_2^2+k_2^2})$ | Goal cannot be achieved | $Z_1$ and $Z_2$ can not be in the same orientation |
| Validity rule 4 | $\arctan(-k_t/j_t) \neq 0$ deg$\pm 20$ deg$(t=1,2)$ | Goal cannot be achieved | $C_V$ must be in one of slots $A$, $B$, $C$ |
| Validity rule 5 | $j_1>0$ OR $j_2>0$ OR $i_1<0$ OR $i_2<0$ | Goal cannot be achieved | Constraints on $\theta_1$, $\theta_2$ and $\theta_3$ |
| Arbitration rule 1 | $\arctan(-k_1/j_1) \neq v_1 \cdot 20$ deg | Perform $a_4[\arctan(-k_1/j_1)/20$ deg$]$ | Relocate $C_V$ from its previous location |
| Arbitration rule 2 | $\theta_1 \neq \arccos(i_1/\sqrt{i_1^2+j_1^2+k_1^2})$ | Perform $a_1[\arccos(i_1/\sqrt{i_1^2+j_1^2+k_1^2})]$ | Move $Z_1$ to a new orientation according to its previous angle. |
| Arbitration rule 3 | $\theta_2 \neq \arccos(i_2/\sqrt{i_2^2+j_2^2+k_2^2})$ | Perform $a_2[\arccos(i_2/\sqrt{i_2^2+j_2^2+k_2^2})]$ | Move $Z_2$ to a new orientation according to its previous angle. |
| Arbitration rule 4 | $\theta_3 \neq \arccos(-k_3/\sqrt{j_3^2+k_3^2})$ | Perform $a_3[\arccos(-k_3/\sqrt{j_3^2+k_3^2})]$ | Move $Z_3$ to a new orientation according to its previous angle. |

performance a set of arbitration rules along with a few validity rules to directly derive a reconfiguration plan have been developed and are shown in Table 1. The application of the plan-searching algorithm in this example is not necessary.

The application program takes the current RMT configuration and the expected cutting directions as inputs, and then consults the rule library to generate a reconfiguration plan. It first evaluates the achievability of the goal through validity rules. Then the program will derive an arbitral plan by firing some of the arbitration rules.

By applying the above rules, some numerical cases are examined. The results are shown in Table 2.

Figure 8 shows how the RMT is reconfigured in case 3. The initial state is (15, 75, 45, 0 deg). The vertical column $C_V$ is relocated from the slot $B$ ($v=0$) to the slot $C$ ($v=-1$). The spindle $Z_1$ is reoriented from the 15 deg angle to the 30 deg angle with $i$-axis, $Z_2$ from the 75 deg angle to the 60 deg angle with $i$-axis, and $Z_3$ the 45 deg angle to the 30 deg angle with $k$-axis. Thus the final state is (30, 60, 30, −1 deg).

Though a simple example, it gives some insight on how an application program interacts with validity and arbitration rules to generate a reconfiguration plan without a plan-searching process. This situation only happens if there is enough knowledge from the human experts to determine a plan for every feasible goal. If arbitration rules cannot solve all cases, the application program should continue with the plan-searching procedure using control rules to find a plan. This will be demonstrated with the next example.

### 5.2 RMS Scalability Reconfiguration Planning

*5.2.1 Description and Execution*. A RMS is a reconfigurable object that comprises a number of member objects such as RMTs. An example is shown in Fig. 1. The system consists of three stages *OP10*, *OP20*, and *OP30*, along with two intermediate buffers *Buf1* and *Buf2*. Two parts *Pt1*, *Pt2* are produced by 3 CNC machines in *OP10*, an RMT in *OP20*, and another 3 CNC machines in *OP30*. Each part contains a number of tasks that need to be allocated to three stages. The processing time of each part in each stage is determined by the tasks allocated to the stage and the number of machines or spindles in that stage.

The RMT being used in *OP20* is a scalable multispindle CNC [17]. This type of RMT combines a standardized CNC base with an adjustable number of spindles for metal cutting tasks. As shown in Fig. 9, the machine can be reconfigured by adding (or removing) spindles, tool changers, and part fixtures. The production rate of the RMT is proportional to the number of spindles. For example, a three-spindles RMT is equivalent to three single-spindle machines. One RMT can mount up to four spindles.

The objective of this example is to generate a reconfiguration plan for the RMS when the production is to be scaled-up, i.e., to increase the throughput of one or both parts. There are two levels of reconfiguration to exploit in scaling the system: machine level and system level. At the machine level, spindles, fixtures, and workpieces to be machined in parallel on the RMT (see Fig. 9) are added to the machine base to scale up production if there are empty slots. System level reconfiguration actions include adding CNC machines, adding a RMT base (if all existing bases have already installed four spindles), increasing buffer size and shifting machines between stages. In this example, shifting of machines can only occur between *OP10* and *OP30* since both use CNC machine tools.

The reconfigurable object model is

$$\text{RMS}=\{\text{Machine: } CNC, RMTBase;\ \text{Spindle: } Spd;\ \text{Stage: } OP10, OP20, OP30;\ \text{Buffer: } Buf1, Buf2;\ \text{Part: } Pt1, Pt2\}$$

The state of the RMS

$$S = \{p1 = (CNC\ OP10\ n_1), p2 = (RMTBase\ OP20\ k), p3 = (Spd\ RMTBase\ n_2), p4 = (CNC\ OP30\ n_3), p5 = (Buf1\ OP10\ b_1),$$

**Table 2 Results of RMT reconfiguration planning**

| Case No | Initial State $(\theta_1, \theta_2, \theta_3, v_1)$ (deg) | Expected performance goal $(i_1, j_1, k_1, i_2, j_2, k_2, i_3, j_3, k_3)$ | Reconfiguration plan (deg) | Rules being fired |
|---|---|---|---|---|
| 1 | (30, 60, 45, 0) | (0.7071, −0.7071, 0, 0.7071, −0.7071, 0, 0, 1, 0) | Empty plan | Violate validity rule 3 |
| 2 | (30, 60, 45, 0) | (0.9659, −0.2588, 0, 0.7071, −0.7071, 0, 0, 1, 0) | $a_2(45)$, $a_3(90)$ | Arbitration rules 2 and 3 |
| 3 | (15, 75, 45, 0) | (0.866, −0.4698, −0.171, 0.5, −0.8138, −0.2962, 0, 0.5774, −1) | $a_4(-1)$, $a_1(30)$, $a_2(60)$, $a_3(30)$ | Arbitration rules 4, 1, 2, 3 |
| 4 | (30, 90, 30, 0) | (1.0, 0, 0, 0.5, −0.8138, 0.2962, 0, 0.7002, −1.0) | $a_4(1)$, $a_1(0)$, $a_2(60)$, $a_3(35)$ | Arbitration rules 4, 1, 2, 3 |

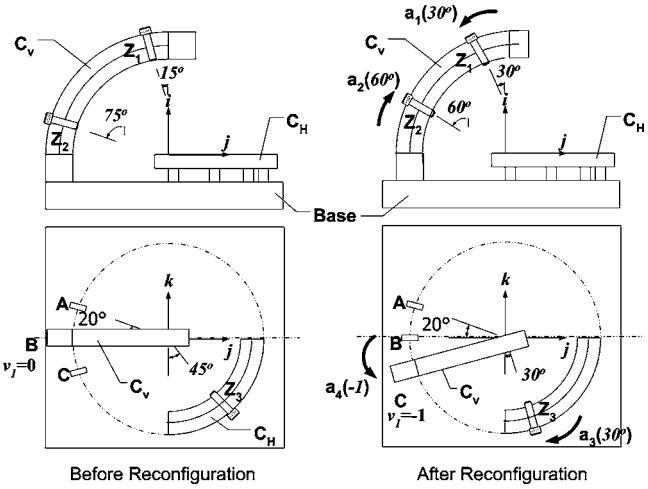


**Fig. 8 Reconfigure the RMT in the case 3**

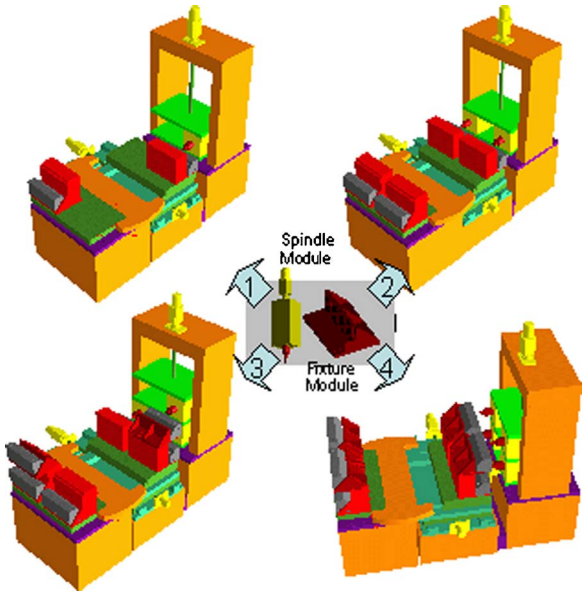


**Fig. 9 Multispindle reconfigurable machine tool**

$$p6 = (Buf2\ OP20\ b_2); n_1, n_2, n_3, k, b_1, b_2, th_1, th_2\}$$

$n_1$ and $n_3$ stand for the number of machines used in *OP10* and *OP30*, respectively. $k$ is the number of RMT bases in *OP20* and $n_2$ is the total number of spindles being installed on RMTs of *OP20*. $b1, b_2$ is the buffer size of *Buf1* and *Buf2*. $th_1$ and $th_2$ are throughputs (parts/minute) of part *Pt1* and *Pt2* with the above system configuration.

The constraints are

$$C = \{q1 = (Buf1\ OP10\ b_1): Buf1 \text{ must be placed after } OP10\}$$
$$\cap \{q2 = (Buf2\ OP20\ b_2): Buf2 \text{ must be placed after } OP20\}$$
$$\cap \{n_1 \leqslant 8, n_2 \leqslant 2, k \leqslant 8, n_3 \leqslant 8, n_1, n_2, n_3$$
$$\in N: \text{Maximal } machine(base \text{ or } spindle) \text{ quantity in each stage}\}$$
$$\cap \{b_1 \leqslant 10, b_2 \leqslant 10, b_1, b_2 \in N: \text{Maximal buffer size is } 10\}$$

The actions and the associated costs are: $a_1$: add one CNC machine to *OP10* for \$300 K; $a_2$: add one spindle to RMT in *OP20* for \$150 K; $a_3$: add one RMT base to *OP20* for \$390 K; $a_4$: add one CNC machine to *OP30* for \$300 K; $a_5$: add one buffer to *Buf1* for \$5 K; $a_6$: add one buffer to *Buf2* for \$5 K; $a_7$: shift one CNC machine from *OP10* to *OP30* for \$12 K; and $a_8$: shift one CNC machine from *OP30* to *OP10* for \$12 K.

The performance metric $Y$ of the RMS is the total production time required to produce the daily demands $D_i$ for each part, which must achieve the goal: equal to or less than 960 min. The assumption of batch production mode in which one part type does not enter into the system until the previous one has reached its daily demand is made

$$Y = (D_1/th_1 + D_2/th_2)$$

Here the throughput $th_i$ is estimated by a throughput analysis software, performance analysis for manufacturing system [18], which utilizes an analytical method to approximate the throughput for each part with consideration of machine reliability and the impact of buffers.

The mapping function is

$$Y = \Phi(n_1, n_2, n_3, b_1, b_2, th_1, th_2, D_1, D_2)$$

The rules being applied to this RMS are shown in Table 3. The validity rules ensure that the goal falls into the upper and lower bounds. Instead of arbitration rules, a number of control rules are defined in order either to specify the constraints on the RMS configuration (rules 1 to 6), or to avoid inefficient action-combination (rules 7 and 8). All rules are equally important and none of them can be violated.

The user manually inputs the rules into the knowledge base and specifies other inputs such as the initial state and goal, task precedence graph, and machine reliability. It only takes several minutes if all information is ready. Totally, 17 cases with from low to high goal demands, which are randomly generated, have been

**Table 3 Rule library of the RMS**

| Rules | IF | THEN | Description |
|---|---|---|---|
| Validity rule 1 | $D_1 < D_{10}$ AND $D_2 < D_{20}$ | No reconfiguration is needed | Both new demands lower than current output |
| Validity rule 2 | $D_1 > D_{1U}$ (upper bound of $D_1$) | Goal cannot be achieved | *Part1* demand exceeds system capacity |
| Validity rule 3 | $D_2 > D_{2U}$ (upper bound of $D_2$) | Goal cannot be achieved | *Part2* demand exceeds system capacity |
| Control rule 1 | $n_1 < 8$ | Perform $a_1$ | *OP10* can add one more CNC |
| Control rule 2 | $n_2 \neq 4 \times k$ AND $n_2 < 8$ | Perform $a_2$ | *OP20* can add one more spindle |
| Control rule 3 | $n_2 = 4$ | Perform $a_3$ | *OP20* can add one RMT base |
| Control rule 4 | $n_3 < 8$ | Perform $a_4$ | *OP30* can add one more CNC |
| Control rule 5 | $B_1 < 10$ | Perform $a_5$ | *Buf1* can add one more buffer |
| Control rule 6 | $B_2 < 10$ | Perform $a_6$ | *Buf2* can add one more buffer |
| Control rule 7 | Not perform $a_8$ AND $n_3 < 8$ | Perform $a_7$ | Actions $a_7$ and $a_8$ are mutually exclusive. Only one of them can be present in a plan. |
| Control rule 8 | Not perform $a_7$ AND $n_1 < 8$ | Perform $a_8$ | |

**Table 4 Results of RMS reconfiguration planning**

| Case No | Daily demands (parts/day) | Reconfiguration plan | Final system configuration | Total production time (min) | Optimal cost (K S) |
|---|---|---|---|---|---|
| 1 | 205/201 | $a_5$ | $n_1=3, n_2=1, k=2, n_3=3$; $b_1=5, b_2=2$ | 958.19 | 5 |
| 2 | 206/202 | $a_6$ | $n_1=3, n_2=1, k=2, n_3=3$; $b_1=4, b_2=3$ | 959.40 | 5 |
| 3 | 206/203 | $a_6$, $a_6$ | $n_1=3, n_2=1, k=2, n_3=3$; $b_1=4, b_2=4$? | 957.55 | 10 |
| 4 | 208/205 | $a_5$, $a_6$, $a_6$, $a_6$, $a_6$ | $n_1=3, n_2=1, k=2, n_3=3$; $b_1=5, b_2=7$ | 959.13 | 25 |
| 5 | 242/201 | $a_2$ | $n_1=3, n_2=1, k=2, n_3=3$; $b_1=4, b_2=2$ | 947.94 | 150 |
| 6 | 245/205 | $a_6$, $a_2$ | $n_1=3, n_2=1, k=2, n_3=3$; $b_1=4, b_2=3$ | 958.95 | 155 |
| 7 | 245/210 | $a_7$, $a_2$ | $n_1=2, n_2=1, k=3, n_3=3$; $b_1=4, b_2=2$ | 954.82 | 162 |
| 8 | 250/210 | $a_6$, $a_7$, $a_2$ | $n_1=2, n_2=1, k=3, n_3=4$; $b_1=4, b_2=3$ | 958.39 | 167 |
| 9 | 250/215 | $a_6$, $a_6$, $a_6$, $a_6$, $a_6$, $a_7$, $a_2$ | $n_1=2, n_2=1, k=3, n_3=4$; $b_1=4, b_2=7$ | 959.81 | 187 |
| 10 | 245/220 | $a_6$, $a_6$, $a_6$, $a_6$, $a_6$, $a_6$, $a_6$, $a_5$, $a_7$, $a_2$ | $n_1=2, n_2=1, k=3, n_3=4$; $b_1=5, b_2=9$ | 959.97 | 202 |
| 11 | 350/325 | $a_2$, $a_7$, $a_2$, $a_6$, $a_6$, $a_6$, $a_5$, $a_6$, $a_6$, $a_4$, $a_3$, $a_2$ | $n_1=2, n_2=2, k=6, n_3=5$; $b_1=5, b_2=7$ | 958.54 | 1182 |
| 12 | 425/500 | $a_2$, $a_2$, $a_6$, $a_5$, $a_6$, $a_6$, $a_3$, $a_2$, $a_4$, $a_6$, $a_4$, $a_2$, $a_4$, $a_2$, $a_4$ | $n_1=3, n_2=2, k=8, n_3=7$; $b_1=5, b_2=6$ | 959.22 | 2365 |
| 13 | 480/500 | $a_2$, $a_2$, $a_1$, $a_3$, $a_2$, $a_1$, $a_2$, $a_2$, $a_1$, $a_4$, $a_1$, $a_4$ | $n_1=7, n_2=2, k=8, n_3=5$; $b_1=4, b_2=2$ | 950.69 | 2940 |
| 14 | 525/520 | $a_2$, $a_2$, $a_4$, $a_6$, $a_6$, $a_6$, $a_4$, $a_3$, $a_2$, $a_2$, $a_4$, $a_4$, $a_1$, $a_1$, $a_1$, $a_4$ | $n_1=6, n_2=2, k=7, n_3=8$; $b_1=4, b_2=5$ | 959.11 | 3405 |
| 15 | 540/550 | $a_2$, $a_2$, $a_1$, $a_6$, $a_6$, $a_6$, $a_6$, $a_6$, $a_5$, $a_6$, $a_5$, $a_6$, $a_5$, $a_6$, $a_5$, $a_5$, $a_3$, $a_5$, $a_2$, $a_2$, $a_1$, $a_2$, $a_1$, $a_4$, $a_1$, $a_4$, $a_1$, $a_4$ | $n_1=8, n_2=2, k=8, n_3=6$; $b_1=10$, $b_2=10$ | 959.86 | 3610 |
| 16 | 580/580 | $a_2$, $a_2$, $a_1$, $a_6$, $a_6$, $a_6$, $a_6$, $a_6$, $a_6$, $a_6$, $a_3$, $a_2$, $a_1$, $a_2$, $a_2$, $a_5$, $a_5$, $a_6$, $a_5$, $a_6$, $a_5$, $a_6$, $a_1$, $a_4$, $a_4$, $a_4$, $a_1$, $a_4$ | $n_1=8, n_2=2, k=8, n_3=7$; $b_1=8$, $b_2=10$ | 959.79 | 3900 |
| 17 | 600/600 | Empty | N/A | N/A | N/A |

examined. The execution time of searching process ranges from 3 s to 20 min, depending on the size of the demands.

*5.2.2 Results*. The results in Table 4 generally show that the higher the daily demands, the more reconfiguration cost is needed to fulfill them. The demands in cases 1–4 are a little higher than the original ones, so minor changes like increasing buffer size and reallocating the tasks are enough. As the demands increase more, it is necessary to expend more money to add extra machines, spindles, or buffers. If the demands are too high as in case 17, the available resources and reconfigurability cannot fulfill the goal and an empty plan is returned.

The optimality of the A* algorithm can be partially justified from the difference between the final performance (total production time) and the goal (960 min). The best solutions should minimize the difference between their throughputs and this production goal. As can be seen from Table 4, most of the cases show small differences (1–3 min). If it is impossible to reach the goal through those actions with lowest reconfiguration cost (such as $a_5$ and $a_6$, adding buffers), the difference becomes larger, as in cases 5, 7, and 13.

Figure 10 shows the output of the application program for case 8 and how the RMS is reconfigured. The machine *CNC13* is moved from *OP10* to *OP30*. One RMT spindle is added to *OP20*. In addition, one extra buffer is added to *Buf2*.

**5.3 Optimality Comparison With Genetic Algorithms Results.** Another plan-searching method based on genetic algorithms (GA) has been designed for comparison with the A* algorithm results. It was felt that this may be more computationally efficient though not as good in finding the optimal solution. In the

```
======= Reconfiguration Planning by A* Searching Algorithm) =======
Throughputs=  250.0,  210.0 (parts/day).

Search found goal state
Displaying solution
Node info: Action 0; 3*Type1, (Buf=4), 2*Spd2, (Buf=2), 3*Type1; RcgCost=   0.0
Node info: Action 6; 3*Type1, (Buf=4), 2*Spd2, (Buf=3), 3*Type1; RcgCost=   5.0
Node info: Action 7; 2*Type1, (Buf=4), 2*Spd2, (Buf=3), 4*Type1; RcgCost=  17.0
Node info: Action 2; 2*Type1, (Buf=4), 3*Spd2, (Buf=3), 4*Type1; RcgCost=  167.0
Solution steps 3
SearchSteps : 330

Task Allocation for Part 1:
Stage 1: 1  3
120.00  60.00  : 180.00 sec
Stage 2: 2  4
120.00 150.00  : 270.00 sec
Stage 3: 5  6
240.00 120.00  : 360.00 sec
Processint Time (Second) in Each Stage:   180.0000          270.0000          360.0000

Estimated Throughput(Demand=  250.00 Parts/Day):  0.5924 Parts/Min =  422.04 Min

Task Allocation for Part 2:
Stage 1: 13
180.00  : 180.00 sec
Stage 2: 14  10  11
150.00 120.00  60.00  : 330.00 sec
Stage 3: 12  9  8  7
150.00 180.00 150.00  90.00  : 570.00 sec
Processint Time (Second) in Each Stage:   180.0000          330.0000          570.0000

Estimated Throughput(Demand=  210.00 Parts/Day):  0.3915 Parts/Min =  536.35 Min

A* Computation Time = 3.1090 Seconds
```

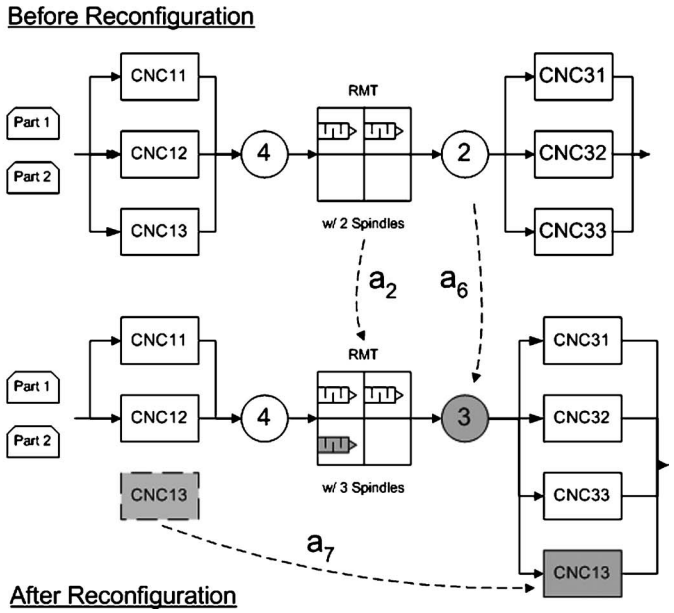


**Fig. 10 Reconfiguring the RMS in case 8**

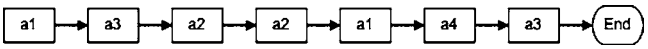


**Fig. 11 A list structure representation of a plan**

GA-based method a reconfiguration plan is represented by a list structure (see Fig. 11) in which each action is a node of the list. With a population of randomly initialized lists, the algorithm applies genetic operations such as reproduction, crossover, and mutation to produce successively better plans in terms of reconfiguration cost while satisfying all rules and constraints.

The GA approach cannot guarantee a global optimum, but it can be run multiple times with varied parameters and the best solution chosen as the final one. The A* algorithm on the other hand can obtain a "real" minimal cost plan as long as its admissibility condition is satisfied. Figure 12 gives the comparison between two methods in terms of minimal reconfiguration cost. When the search space is small (cases 1–13), both methods find the optimal solution. As the search space gets larger as with cases 14–16, the A* method gets better results than the GA method though both values are very close. Usually the difference is a buffer cost (action $a_5/a_6$) or machine exchange cost (action $a_7/a_8$). To some extent, this validates the optimality of the A* algorithm.

Although the GA method obtains inferior results in large search spaces, it takes far less computation time to find a good result than the A* does. As shown in Fig. 13, the GA uses approximately the same amount of time although the search space grows larger. In contrast, the computation time of the A* is heavily impacted by the complexity of the search space. Considering optimality and efficiency factors, the GA can be a good alternative methodology to the A* planner in large-space problems. Further studies on a GA planner will be conducted on the basis of this paper.

## 6 Future Directions

The A* plan-searching algorithm can only derive a linear plan which consists of sequential reconfiguration actions. For complex reconfigurable objects, a *nonlinear reconfiguration plan* allowing parallel actions is necessary to save time-dependent resources like operational and labor costs. A revised heuristic function may be a solution in this case. The cost of the path should not only include the reconfiguration cost but also the actual execution time for actions. The actual time can be transformed into the unit of cost by applying appropriate cost rates. A weighted sum of the cost and time can then replace the cost currently used in the heuristic function.

Some possible enhancements could add to the A* algorithm to reduce the computational efforts. A preprocess can be performed to determine the maximum number of units needed for each action to satisfy the new requirements. Besides, the redundant path (equivalent action combinations) should be singled out during the searching process.

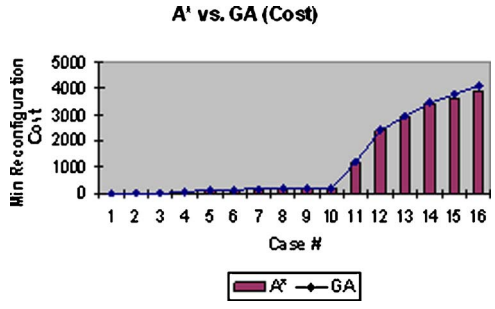


**Fig. 12 Optimality comparison between A* and GA**

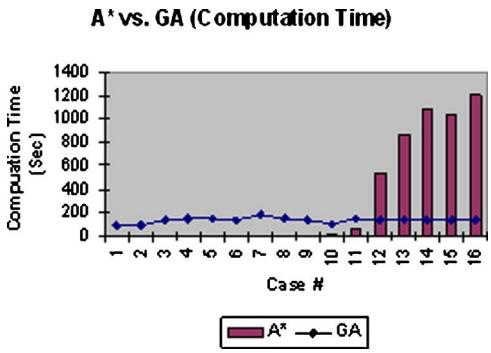


**Fig. 13 Computation time comparison between A* and GA**

Extracting reconfigurability from existing data is another issue to be investigated. An automated or semi-automated method will expedite the process of building a reconfigurable object model. Furthermore it will be necessary to pursue research into related fields such as CAD assembly modeling and virtual factory modeling in order to expand capabilities in these areas to capture and manage reconfiguration data.

More practical examples are under way to test the performance of the algorithm with a larger search space. The GA method could be one way to save computation time while obtaining a very close-to-optimal solution. On the other hand, it may be necessary to investigate A* method variants such as the iterative-deepening A* to reduce memory requirements [19], since the memory used by the A* algorithm grows exponentially with the depth of the goal in the search space,

## 7 Conclusions

An AI-based CARP framework has been developed in order to derive reconfiguration plans for a RMS and reconfigurable hardware in the system. A generic reconfigurable object model is presented to capture necessary information for all levels of objects in the RMS. Based on this model, the framework either generates a plan directly according to arbitration rules or finds a plan that requires minimal resources with the control rules and constraints applied within an A* plan-searching framework. Case studies in planning a RMT and a RMS are conducted and the results show that efficient plans are generated in both situations.

## Acknowledgment

The authors are pleased to acknowledge support of this research by the National Science Foundation Engineering Research Center for Reconfigurable Manufacturing Systems under Grant No. EEC-9529125.

## References

[1] Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritchow, G., Van Brussel, H., and Ulsoy, A. G., 1999, "Reconfigurable Manufacturing Systems," CIRP Ann., **48**(2), pp. 527–540.
[2] Askin, G. R., and Zhou, M., 1998, "Formation of Independent Flow-Line Cells Based on Operation Requirements and Machine Capabilities," IIE Trans., **30**, pp. 319–329.
[3] Kimms, A., 2000, "Minimal Investment Budgets for Flow Line Configuration," IIE Trans., **32**, pp. 287–298.
[4] Donohue, K. L., Hopp, Wallace J., and Spearman, M. L., 2002, "Optimal Design of Stochastic Production Lines: A Dynamic Programming Approach," IIE Trans., **34**, pp. 891–903.
[5] Tang, L., Yip-Hoi, D. M., Wang, W., and Koren, Y., 2003, "Concurrent Line-Balancing, Equipment Selection and Throughput Analysis for Multi-Part Optimal Line Design," *CIRP 2nd International Conference on Reconfigurable Manufacturing*, Ann Arbor, MI.
[6] Bradford, J., and Childe, S. J., 2002, "A Non-Linear Redesign Methodology for Manufacturing Systems in SMEs," Comput Ind., **49**(1), pp. 9–23.
[7] Aiken, P., and Hodgson, L., 1998, "Synergy Between Business Process and Systems Reengineering," Information Systems Management, **15**(4), pp. 55–67.
[8] Chan, F. T. S., and Jiang, B., 2001, "The Applications of Flexible Manufac-

turing Technologies in Business Process Reengineering," Int. J. Flexible Manuf. Syst., **13**, pp. 131–144.
[9] Moon, Y.-M., and Kota, S., 1998, "Generalized Kinematic Modeling Method for Reconfigurable Machine Tools," *Proceedings of ASME Design Engineering Technical Conference*, Atlanta, GA.
[10] Son, S.-Y., Olsen, T. L., and Yip-Hoi, D., 2000, "A Genetic Algorithm Approach for the Design of Machining System Families," *Proceedings of the 2000 International CIRP Design Seminar, Design With Manufacturing: Intelligent Design Concepts, Methods and Algorithms*, Haifa, Israel.
[11] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., 2001, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA.
[12] Dechter, R., and Pearl, J., 1985, "Generalized Best-First Search Strategies and the Optimality of A*," J. Assoc. Comput. Mach., **32**(3), pp. 505–536.
[13] Cherif, M. a. G. K. K., 1999, "Planning Quasi-Static Fingertip Manipulations for Reconfiguring Objects," IEEE Trans. Rob. Autom., **15**(5), pp. 837–848.
[14] Nilsson, N. J., 1998, *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann Publishers, San Francisco, CA.
[15] Du, H., and Lin, G. C, I., 1998, "Development of an Automated Flexible Fixture for Planar Objects," Rob. Comput.-Integr. Manufact., **14**(3), pp. 173–183.
[16] CLIPS: A Tool for Building Expert Systems, http://www.ghg.net/clips/CLIPS.html.
[17] Spicer, P., Koren, Y., Shpitalni, M., and Yip-Hoi, D., 2002, "Design Principles for Machining System Configuration," CIRP Ann., **51**(1), pp. 275–278.
[18] Yang, S., Wu, C., and Hu, S. J., 2000, "Modeling and Analysis of Multi-Stage Transfer Lines With Unreliable Machines and Finite Buffers," Ann. Operat. Res., **93**, pp. 405–421.
[19] Russell, S. J., and Norvig, P., 1995, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Inc., Englewood Cliffs, NJ.