Artif Intell Rev (2012) 38:67–83
DOI 10.1007/s10462-011-9241-y

A robot decision making framework using constraint programming

Richard S. Stansbury · Arvin Agah

Published online: 28 May 2011

Abstract An intelligent robotic system must be capable of making the best decision at any given moment. The criteria for which task is "best" can be derived by performance metrics as well as the ability for it to satisfy all constraints upon the robot and its mission. Constraints may exist based on safety, reliability, accuracy, etc. This paper presents a decision framework capable of assisting a robotic system to select a task that satisfies all constraints as well as is optimized based upon one or more performance criteria. The framework models this decision process as a constraint satisfaction problem using techniques and algorithms from constraint programming and constraint optimization in order to provide a solution in real-time. This paper presents this framework and initial results provided through two demonstrations. The first utilizes simulation to provide an initial proof of concept, and the second, a security robot demonstration, is performed using a physical robot.

Keywords Constraint programming · Mobile robots · Applied artificial intelligence

1 Introduction

An important requirement of all robotic systems is the ability to rationally choose the "best" task to execute at any moment of its operation. Often, constraints upon the system limit which tasks are available of those from which the robot can choose. Constraints exist based on safety, accuracy, mission needs, performance, robot capabilities, etc. In addition to satisfying constraints, the best solution available would be one that also maximizes some utility.

R. S. Stansbury (✉)
Department of Electrical, Computer, Software, and Systems Engineering, Embry-Riddle Aeronautical University, Daytona Beach, FL 32114, USA
e-mail: stansbur@erau.edu

A. Agah
Department of Electrical Engineering and Computer Science, The University of Kansas, Lawrence, KS 66045, USA
e-mail: agah@ku.edu

This paper presents a new framework for task selection in which the decision making logic is modeled as a constraint satisfaction problem (CSP) using constraint programming (CP) and constraint optimization (CO) tools.

The constraint satisfaction problem is a research domain from the Artificial Intelligence community. A problem is comprised of a set of variables, each variable's domain of available assignments, and constraints. Constraints define the valid assignment of values between one or more variables within the model. They are capable of defining very simple rules similar to those seen in rule-based systems to complex relations between variables necessary to bind the solution based on the problem domain.

CSPs have been applied to many application areas include scheduling, design, and image processing. Their utilization within the domain of robotics has been historically very limited due to their computational complexity (NP-Complete). As algorithms and heuristics for solving CSPs improve, CPU speeds increase, and software on-a-chip techniques become more prevalent, real-time constraint satisfaction will be possible within the next decade.

This paper presents a new framework for task selection and configuration that models the decision making as a constraint satisfaction problem. It begins with some relevant background regarding constraint satisfaction problems and constraint programming. Next, the framework for task selection is revealed. A simulation-based proof of concept experiment is next presented for an antarctic field robot. A concrete demonstration is then presented in which physical robots utilize the framework in which a robot must act as a sentry and choose the best way to define a restricted area. The paper then concludes with some discussion of these initial results and future work.

2 Background

This section discusses the background of this research approach including: constraint satisfaction problems and constraint programming.

2.1 Constraint satisfaction problems

The constraint satisfaction problem (CSP) is a problem solving technique in Artificial Intelligence (Russel and Norvig 2002). A CSP is defined as a set of variables, each variable's domain, and the constraints upon the variables. Constraints define the valid assignment of values between one or more variables. A CSP is solved when every variable is assigned a value such that no constraints are violated. A problem may have more than one solution. If a utility method is available, the solution that maximizes the utility function is the optimal solution (Russel and Norvig 2002). Algorithms for solving or optimizing constraint satisfaction problems have been presented in a number of textbooks and journal articles including (Dechter 1991; Kumar 1992; Tsang 1993; Mackworth 1997a).

CSPs provides a domain-independent representation of problems such that general search techniques can be used to solve the problem. These search algorithms may involve a global search for a solution, or a local search in which a valid solution is obtained by revising a temporary and invalid solution until a satisfactory solution is reached (Russel and Norvig 2002).

The constraint satisfaction problem approach has been used to solve several real-world problems. Scheduling and planning are common application domains for which CSP techniques are applied. A few areas in which such systems have been applied include university course timetabling (Deris et al. 1997), scheduling tasks for manufacturing processes

(Syrjanen 1998), and scheduling resources for a trucking company (Chun and Wong 2004). Airline scheduling is another common application of CSPs (Russel and Norvig 2002).

2.2 Constraint programming

Constraint programming provides a means of specifying constraint satisfaction problems (Bartak 1999). Constraint programming is often declarative in which a system is modeled using either a special language or a software library that is often accompanied with a CSP solver. These constraint programming environments support a set of constraints that are commonly encountered when specifying a constraint satisfaction problem.

Constraint programming has been implemented in several environments using a variety of languages and methodologies. Some constraint programming environments use their own custom language to define the problem. Others environments provide software APIs written in a variety of common programming languages (Java, C/C++, LISP, etc) (Bartak 1999). ILOG (IBM 2009) and Koalog Java Constraint Solver (Georget 2009) are both commercially available libraries/engines for solving constraint satisfaction problems using constraint programming techniques.

3 Related work

As this research focuses upon the application of constraint satisfaction problems for robot control and decision making, a literature survey was performed to look at previous examples of CSPs entering the robotics domain. ThingLab (Borning 1981) was developed in the early 1980s as a software package that allowed users to graphically design and simulate a system that utilized constraints to define the properties of the system and guided simulation of circuits, mechanical linkages, and other geometries.

D. K. Pai at Cornell University proposed a framework for robot programming using constraint satisfaction techniques (Pai 1989, 1991). Linear inequality constraints were used to define the relationship between mechanical components within the system. Pai defined two special classes of CSP variable. Input variables were used to capture current state and output variables were defined such that their values can be taken from the solution and translated to vehicle actuator commands. For each iteration, the previous solutions were fed into the solver, and then the input variables were updated. A repair algorithm then fixed the solution given the changes and the constraints.

At the University of British Columbia, Zhang and Mackworth developed a constraint-based system for the design of intelligent agents (Mackworth 1997b; Zhang and Mackworth 1994, 2005). The system was designed to support hybrid agents i.e., agents with both continuous and discrete interactions with their environment. A constraint satisfying controller could be synthesized by driving the system toward satisfying all constraints. During operation, if the robotic system deviated from a satisfactory state, the controller would act to drive it back toward a satisfying state (Mackworth 1997b; Zhang and Mackworth 1994, 2005).

4 Decision making framework

This research focuses upon the use of constraint programming as a tool for modeling robot decision making for task selection and configuration. It does not focus upon the algorithms to solve constraint satisfaction problems. The Koalog Java Constraint Solver is only a tool

used to support this project by providing a stable constraint programming and constraint optimization environment (using a constructive search algorithm).

In this section, the constraint-based framework for robot task selection and optimization shall be presented. First, the components and structure of the task models is defined. Next, the techniques involved with constructing the mission-level task selection process is discussed including the merging of constraint models, the inclusion of mission level constraints, and the utilization of performance objectives. Finally, the cyclic process of build, solve/optimize, and execute are defined.

4.1 Linguistic data representation

A CSP algorithm's efficiency is dependent upon the size and shape of the search space. Reduction of the variable domains within the CSP will reduce the solution time. To address this issue, all data used within the framework is linguistically specified or enumerated rather than given a continuous integer domain. Most often a set of five linguistic values is used: {LOW, MED_LOW, MED, MED_HIGH, HIGH}, which are enumerated $\{-2, -1, 0, 1, 2\}$. For most variables in our models, this is the sparse domain used. For the experiments provided, this domain size was sufficient; however, for future work, it is perfectly reasonable to define other linguistic domains such as for heading, distance, duration, etc.

For this research, the conversion for continuous-to-linguistic and linguistic-to-continuous is handled through a set of rules. To take a continuous sensor value to a linguistic value, thresholds are used to determine under which linguistic descriptor it should fall. To take a solution parameter and convert it to a robot specific value, a similar mapping is made for each. This does place some burden on those creating the constraint model to identify the performance of the robot (sensing and actuation) and derive these mappings.

4.2 Task models

For each possible task the robot may execute, a constraint model must be defined. The constraints within the model define the relationship between the robot's state and whether or not the task is an acceptable output. If acceptable, the model will also define the relationship between the state of the robot and the configuration of the task's execution (i.e. if the robot can execute the task, are there any parameters such as speed regarding how it is executed?). Given this configuration, constraints also define the level of acceptability of this solution as some configurations are more desirable than others.

4.3 Variables

The constraints are defined upon variables. There are four types of variables that have been defined as part of this research: input, output, hidden, and performance. To maintain a reasonable problem size, each variable is discretized and linguistic values as described above. Input variables are constrained at run-time to equal the current state of the robot. Output variables define the execution parameters of the task as discussed above. Hidden variables are used to simplify constraint declaration by combining one or multiple input variables into a single internal variable. Each task may have one or multiple performance variables (i.e. quality, timeliness, etc.) and a hidden performance variable such that constraints may tie multiple performance variables into a single value. For each task, at runtime, a unique identifier is given to each task, which is treated as an input variable. These terms and special variable

types are used to define the model, but are transparent to any CSP search or optimization algorithm.

4.4 Constraints

The remainder of the constraint model is comprised of the constraints themselves. Each task may have zero, one, or multiple constraints. The constraints limit the domain of an output variable given the current state of input and/or hidden variable(s). Constraints may also tie the domain of the performance variable to value(s) of input, output, and hidden variable(s).

4.5 Extending task models

As part of the implementation of this task model framework, it was determined that it would be highly desirable to extend" a task model similar to the extension of classes in object-oriented programming. Consider a "move-to" task in which the robot must go to a location at a predefined speed maintaining a minimum safe distance from obstacles. A designer could define must of the task model without any knowledge of the target robot platform. However, once a platform is selected, additional constraints may need to be added. Under the current framework, it is possible to create a new constraint model that extends an existing model by adding new constraints and/or variables to the model.

4.6 Task selection model

Each individual task model, once solved, produces three distinct products. First, if a solution may be derived, then it may be concluded that it would be valid for the task to be executed under the solution parameters. Second, it provides the necessary configuration of parameters for the task such that when it executes it will satisfy all of the existing constraints. Third, if the model is optimized, the solution will also provide an indication of how "optimal" a solution is based on the defined performance metric.

Rather than determine if just one task is valid, it is the goal of this research to be able to determine which task among a set of possible tasks may be executed and how it must be executed such that it is the best task to execute at the moment and such that it meets all task and mission constraints. To accomplish this goal, a task selection model must be generated that is composed of the set of task models from all possible tasks that may be considered. Mission constraints and any additional mission level variables may also be added to the task selection model to provide any global constraints upon the system. The optimization of such model should provide the single best solution.

Since the optimizer only attempts to maximize a single variable, constraints must tie each task's performance variable to a single, mission-level performance variable. Therefore, if the solution picks a particular task, the mission-level performance variable will store the performance of the selected task.

Construction of the task selection module is relatively straight forward as it builds upon some common principles of constraint programming. Each model is composed of the constraints and the variables. If optimizing the model, one of the variables must be designated as the optimization variable. So long as each model has unique variables, it is possible to create a new model by merging the set of variables and the set of constraints into a single problem. This merge is similar to the "extend" operation described above. If mission level variables and constraints exist, they can be added to those two sets. A constraint solver will not care about where the variables and constraints originated.

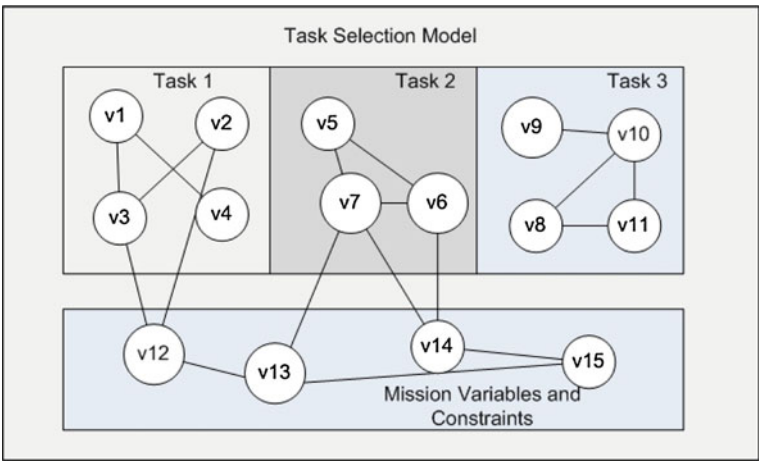

Fig. 1 Composition of task selection model from multiple task models (in the graph variables are nodes and constraints are arcs)

Figure 1 depicts the task selection model as composed of its constituent parts: variables and constraints. The circles represent the possible variables within each model. The arcs represent the constraints between variables. For global variables related to the entire mission such as percepts, only one instance of each variable exists and they are tied into each of the task models through constraints.

Once the model is constructed, a constraint satisfaction problem optimization algorithm produces a single solution in which a near optimal solution is generated. Given model size, it may be too time consuming to find the global optimal solution. Many constraint optimization solvers operate such that once an initial solution is found it will iteratively work to improve the solution. Thus, if the algorithm is preempted due to a time limit, if a solution exists, it will satisfy the problem.

4.7 Task selection process: model-optimize-execute

Given the task models and the process for creating the task selection model, let's now consider the complete process of modeling tasks, optimizing for a solution, and executing the given solution. It should be noted that the process executes asynchronously and is triggered to begin at the completion or failure of the robot's current task.

The modeling phase involves the updating of the model to reflect the current state of the robot. For this phase, constraints are applied to the input variables of each model such that their domains are constrained to the current state of the robot. Next, the task selection model is constructed through the merge process discussed above.

The optimize phase involves the use of our constraint optimization algorithms. The optimizer is configured such that it has the current model and knows which variable to optimize within the model. A time limit is set for the optimization algorithm and the algorithm begins. The algorithm executes until either a global maxima is found or the time limit is reached. The model's solution is returned, but not yet in a usable form.

The execute process begins with the decomposition of the solution. The solution is analyzed and the "winning" task is determined. Then, the execution parameters for the task are obtained from the solution. Finally, the task is executed given the execution parameters. The

robot will carry out the task until the task either reaches some ending success for failure state. Once the task execution ends, this cycle repeats itself.

5 Simulated proof-of-concept

Prior to implementing a constraint-based decision maker for a physical robot, a simulated scenario for a mobile robot operating in Antarctica called the "Polar Robot Scenario" provides a proof-of-concept (Stansbury 2007; Stansbury and Agah 2008).

5.1 Polar robot scenario

Polar research activities benefit from the use of unmanned systems including ground and aerial robots. Remote sensing tasks in these environments require tedious or unsafe routes, highly precise movement, and/or continuous long-term operation. A single mobile robot can be equipped with a number of remote sensing instruments and a variety of power generation sources (Stansbury et al. 2004; Lever et al. 2006).

Each sensor, each power generation source (solar, gas generator, etc), and the robotic platform's unique operating constraints. Likewise, the mission has its own constraints upon how the data must be collected. Constraint-based task selection provides a valid means for determining which task to perform and the configuration of the robotic system to perform that task.

5.1.1 Robot and environment

The polar robot is simulated based on existing polar robot data such as Dartmouth's Cool Robot (Ray et al. 2005) and the University of Kansas's PRISM mobile robot (Akers et al. 2006). The simulated robot is equipped with sensors that provide percepts of its state. It is also equipped with a remote sensing payload: synthetic aperture radar (SAR), accumulation radar, magnetometer, gravimeter, and IR spectrometer. Each sensor is simulated based on the specifications of commercial off-the-shelf (COTS) components or from experimental systems (Gogineni et al. 2003). Power sources were also simulated including: primary battery, backup battery, wind turbine generator, and deployable solar array. Each power source is simulated based on specifications from COTS components.

The polar environment is simulated as accurately as possible to recreate some of the long-term survivability challenges. As the robot carries out its activities, the time and current date are incremented. Monthly weather date and the simulation's current month provide the data necessary to simulate the current weather conditions. For each month, an almanac stores annual maximum and minimum measurements for climate condition including wind speed, temperature, and solar irradiance (Center for Astrophysical Research in Antarctica 2009). Using a pseudorandom number generator, every 24 h, a current value for each condition is selected from between the minimum and maximum for the current month. Terrain conditions such as hard smooth surfaces, rugged surfaces, and software surfaces are selected based on the current wind speeds and temperature.

The robot's operation is defined such that it may draw power from its primary battery and one additional alternative power source. Only one remote sensing instrument can be deployed at a given time. The robot may remain stationary to charge or may traverse the ice sheet collecting data. Damage or component failure can occur if operating a sensor or power source outside of the manufacturer's specified operating range (wind loads, temperature, vibration, etc).

5.1.2 Task models

Briefly, the variables and constraints of each of the task's models for the polar scenario are presented.

Constraint variables Task constraints are defined in terms of performance, input, and task parameter variables. Performance variables include *TaskRiskAvoidance*, *TaskResourceConservation*, *TaskTimeliness*, and *TaskQuality*, which define from Low to High the level of achievement of each within the solution. Input variables are variables that define linguistically the environmental percepts (e.g. wind speed, solar energy, etc) and the proprioceptive percepts (e.g. battery level) derived from the robot's current internal state (e.g. battery level). Execution variables define generator selected, instrument selected, drive speed, and heater level. Numerous hidden variables are utilized to define intermediate variables between constraints.

Charge task The charge task model defines the constraints for selection of a power source, which is used to charge the robot for 12 consecutive hours. The task's performance is defined by quality, which is equal to the anticipated output level of the power source. The constraints ensure that the robot enters only safe and valid hardware states when using the power generators to avoid damage. The charge task executes by deploying the selected generator and waiting for 12 h in order to collect energy. If no generator is selected, the robot would waits for conditions to change for 12 h before selecting another task.

Survey task Prior to operation, the robot receives one or more survey task requests. Each survey task is assigned at instantiation an instrument, minimum survey quality, and a total survey distance. The survey task defines constraints in several categories: drive, remote sensing instrument, and performance constraints. All constraints regarding power sources from the charge task are also re-used. Drive constraints define the level of risk, timeliness, and power usage of the drive system given the drive speed and the current terrain conditions (rough, smooth, or soft). Remote sensing constraints are defined for each of the instruments. These constraints tie the quality of the survey using an instrument to the input and output variables of the task. For instance, the SAR is constrained such that if the terrain was rough, the data quality was less than medium. The execution of the survey task involves the following procedure. First, the instruments and generators are activated and/or deployed. The robot traverses the ice at the selected speed. Every 1,800 s, the robot determines if it has completed the traverse or may continue onward. Once the target distance is reached, or 24 h had elapsed, the robot terminated its traverse retracting both the power generators and remote sensing instruments. If the survey is completed, the executed survey task was removed from the pool of possible tasks. Otherwise, the task remains available for execution in the future.

5.1.3 Experimental setup

To evaluate the system, a simulated robot and environment were created. At the beginning of an experimental run, the robot is assigned a set of tasks. The robot is asked to perform the tasks and its performance is measured using three metrics described below. The experiment terminates if one year has elapsed or if the robot becomes inoperable to due to damage or lack of power.

Table 1 presents six configurations used to test the decision makers for the polar robot. Each configuration is given an identifier so that it may be easily identified in the tables

Table 1 Polar scenario configurations for evaluation

ID	Number of tasks	Survey distance	Instrument distribution	Performance
POLAR1	5	1,000	Uniform	Default
POLAR2	5	1,000	Uniform	Resources = MED_HIGH
POLAR3	5	1,000	Uniform	Quality = Random
POLAR4	5	1,000	Random	Default
POLAR5	10	500	Uniform	Default
POLAR6	10	500	Random	Default

provided below. The number of survey tasks is specified in the second column. The third column defines the distance that must be traversed for each of the assigned tasks. The instrument distribution column defines how remote sensing instruments are allocated for the set of assigned tasks. The first option is uniform where there would be an equal number of tasks assigned with each instrument. The second option is random in which for each task an instrument is assigned to it pseudo-randomly. The final column, performance, defines the performance requirements set forth upon the robot when selecting and executing its task. The default performance configuration is for medium to be assigned for each of the performance variables. Each receives an equal weighting. For experiment POLAR2, each of the performance variables is set to medium except resource conservation, which is set to medium high. For experiment POLAR3, each of the performance variables is set to medium except quality which was set to random. The quality of collecting data for each task is randomly set.

A rule-based task selector was implemented in order to compare the constraint-based system versus a reasonable alternative. In order to keep the length of this article within the guidelines, the implementation of the rule-based system is not described herein, but may be viewed fully in (Stansbury 2007).

Three metrics were used when comparing the rule-based and the constraint-based systems for each configuration. *Survival time* was defined as the number of days in which the robot survives during the simulation with a maximum of 366 days (leap year). *Failure rate* was the percentage of runs for each in which the robot's operation is terminated due to failure before the 366 days have elapsed. It is expected that the failure rate will be high as the robot has been given more tasks and longer distances that would be feasible for it to complete over a year. A task is classified as a failure if the robot is unable to complete the entire traverse distance with the remote sensing instrument active. This paper will therefore compare the performance of both systems to see which has a smaller failure rate. *Mission completeness* was calculated as the distance surveyed by the robot during the simulation versus the total distance requested for all tasks.

5.2 Results and discussion

Given the metrics, the following results were obtained:

Failure rate is presented in Fig. 2: Polar scenario: failure rate versus configuration (Fig. 2). For the rule-based decision maker, the rate was near 100% meaning that it was unable to

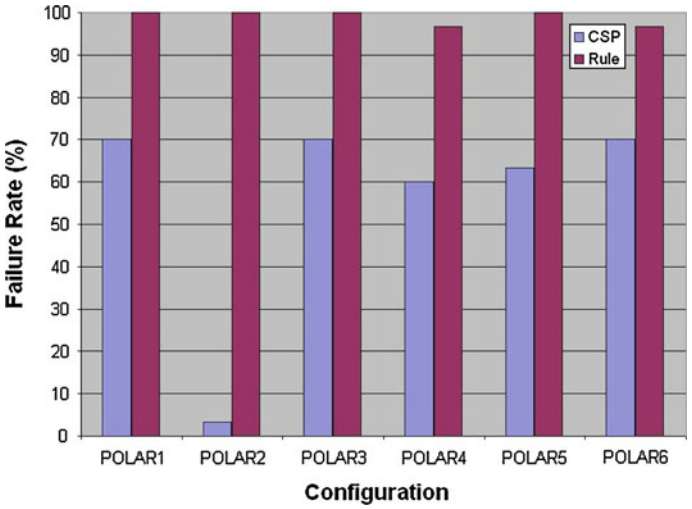

Fig. 2 Polar scenario: failure rate versus configuration

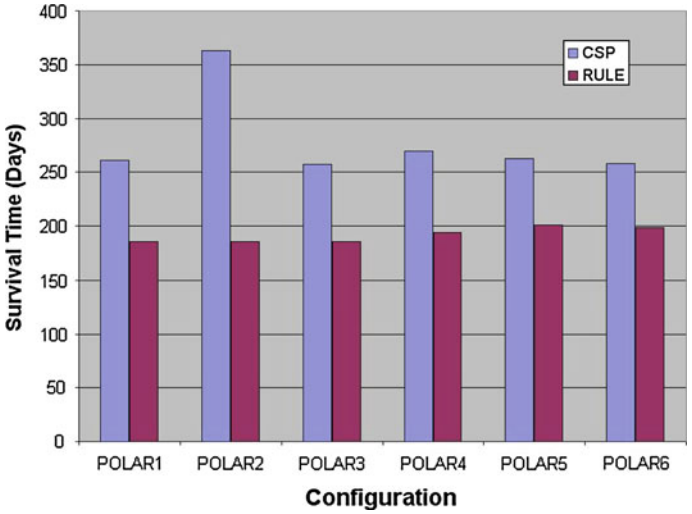

Fig. 3 Polar scenario: survival time versus configuration

choose tasks sufficiently such that the robot was capable of completing all measurements before either the year was complete or the robot was unable to sustain itself. Using constraints, the maximum is near 70%. The constraint-based decision maker outperformed the rule-based under all configurations by surviving much longer.

Survival time is presented in Fig. 3. The rule-based system under all scenarios failed near the half-year mark during the Antarctic winter (harshest climate conditions). The constraint-based decision maker outperformed the rule-based.

Mission completeness is presented in Fig. 4. Using constraints, the mean mission completeness for configurations POLAR1, POLAR3, POLAR5, and POLAR6 was, statistically, significantly greater with means greater than 40%, compared to the mean when using rule-

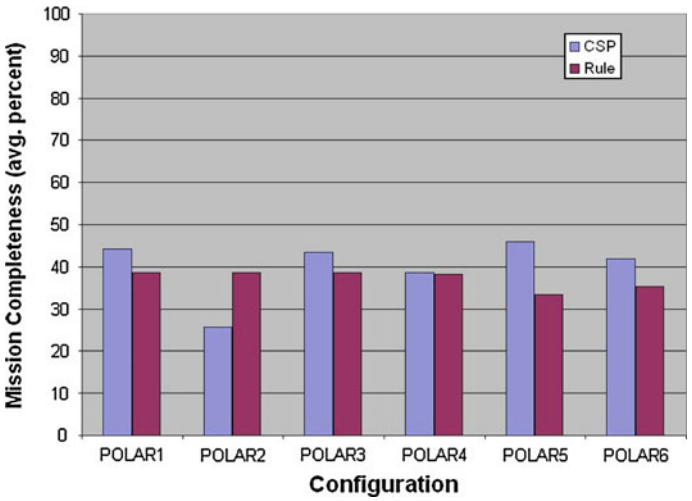

Fig. 4 Polar scenario: completeness of total mission versus configuration

based which were less than 40%. The POLAR4 results were too close to be conclusive. POLAR2 performed much worse than the competing system as it was over-constrained toward conserving resources.

6 Sentry robot demonstration

The sentry robot demonstration requires a mobile robot to defend a "restricted" area from potential invaders. This scenario was originally envisioned for areas in which wildlife must be kept out of an area such as near an airport. While monitoring the area, the robot could stand still guarding a particular direction, go on patrol, scan a 360 degree area from a fixed position, etc. If a threat is detected, the robot has several options. It could consider the threat to be insufficient to warrant a response and continue monitoring, or it could choose to actively engage the target chasing it out of the area. The role of the sentry is to minimize the amount of time in which an intruder is within the restricted zone. The robot must actively attempt to prevent the entry into the area. Should entry occur, it is the robot's role to expel the invader from the restricted area, but not to follow it outside of its designated mission area.

For this scenario, a simple environment was devised in which the experiments are performed. The environment is flat with two designated areas. An outer rectangular perimeter is denoted the mission environment for the sentry. It may only operate within those bounds. Within the outer perimeter, a second perimeter denotes as the restricted area. The center of the restricted area is denoted as the robot's *home* location. The operating area is divided into an occupancy grid, which stores locations where the robot's sonar located potential threats. The variable *Threat Level* is computed by the total number of locations in which a threat may exist given current sensor data.

6.1 Task models

In this section, the abstract task models for the Sentry robot are defined.

6.1.1 Chase

The Chase task directs the sentry robot to move toward and then chase a target. It is expected for all detected targets (potential intruders) that there exists an instance of the Chase task model for that target. Upon selection and execution of a chase task instance, the robot shall chase the target until either a timeout occurs or the target has traveled sufficiently far enough away from the restricted area such that the sentry may break off pursuit.

For the task model, a number of input variables define the relationship of the robot with its environment and its relationship with the target. The *currentThreatLevel* and *averageThreatLevel* are input variables that are merged into a single hidden variable *threatLevel*. The distance of the target from the center of the restricted area is provided as an input variable, *targetDistanceToHome*, which may be interpreted as the level of threat generated by the particular target. Since targets move and noisy data can result in the detection of false targets, *targetConfidence* provides an indication of how confident the system is that the target actually exists. Lastly, *powerRemaining* indicates the level of power remaining within the robot platform. Two output variables exist: *range* and *duration*. *Range* defines the maximum distance and *duration* defines the maximum time duration for which the chase may persist, respectively.

The constraints of the chase are based upon the level of threat and the distance that the robot must leave its home location. The further the robot is away from the center of the restricted area, the less capable it is of handling other threats that may emerge during the case. For the chase task, as the threat level increases, the duration of the chase is decreased. These constraints exist as it is desirable to restrict the time spent away from the center of the restricted area where the sentry is more strategically positioned. If the power is low or medium low, the duration of the scan is reduced to being less than medium in order to avoid expending too much charge during the chase. The range of the robot is constrained based on the target's distance to the center of the restricted area. The further the target is away from the center of the restricted area the less travel distance is permitted in the case.

6.1.2 Scan

The Scan task guides the robot to spin in place generating a 360 degree scan of the area for threat detection and tracking. At the center of the restricted area, the robot will scan counter clockwise at a specified rate. The scan persists until either a threat is detected within some threshold distance or the timeout duration is reached.

Input variables are based on the current and average threat level of the environment. The maximum of these two variables is stored in a hidden variable, *threatLevel*. The *powerRemaining* variable defines the power available for the robot.

The output variables for this model are turn speed, threshold distance, and timeout duration respectively. The constraints in this model choose a timeout and threshold based on the threat level. As the threat level increases, both the timeout and the threshold decrease. The turn speed increases as the threat level increases as it seems to more actively track the increasing number of threats.

For this task, the threshold of the scan is constrained based upon the current threat level. As the threat level increases, the threshold is constrained to be lowered to focus on the area of more immediate need. As the threat level increases, the turn speed also increases such that if the threat level is high, the scan must also be high so that the robot is better at tracking of all targets surrounding it. The duration is constrained to the power remaining. As the power

remaining diminishes, the commitment of time to the scan is decreased to avoid wasting power.

6.1.3 Patrol

The Patrol task attempts to overcome the limitation of the scan task; namely, the limited range for which it is capable of scanning. The robot while patrolling will follow a rectangular path defined by a length and width centered about the home location. The speed of the robot following the path is constrained to the path size. The patrol continues until the time duration is reached or a threat is detected.

Input variables are based on the current and average threat level of the environment. The maximum of these two variables is stored in a hidden variable, *threatLevel*. Another variable stores the current power/fuel remaining onboard the robot.

The output variables define a number of parameters. The *pathLength* and *pathWidth* variables define the relative length and width of the patrol path. A hidden variable, *pathArea*, is used for a variety of constraints within the task model equal to the product of the *pathLength* and *pathWidth*. Other output variables include the *speed* variable, the *duration* variable, and the *threshold* variable (all related to the parameters discussed above).

A number of constraints define the model. If the threat level is greater than or equal to medium, the duration of the search is less than or equal to medium high. If the threat level is high, then the duration is less than medium. This ensures that the robot will not spend too much time scanning if a large number of threats are nearby. As the threat level increases, the threshold is reduced in order to focus on more immediate threats and the patrol area is reduced to keep the robot closer to the center of the restricted area. Given a desire for a consistent orbital rate, as the size of the patrol area increases, the target speed of the robot is constrained to increase. Lastly, the duration is reduced if the power remaining is less than medium in order to ensure that the robot does not run out of charge spending its time on patrol.

6.1.4 Retreat

The Retreat task forces the robot to travel from its current location to the home location a predefined location (most often located at the center of the "restricted" area). This task is included primarily to act as a default task to be selected when no other suitable task exist (i.e. given the constraint model, no other satisfactory solutions exist).

6.2 Experimental setup

An AmigoBot from Mobile Robots Inc (2009). was used as the sentry robot for the demonstration. It possesses six sonar sensors (four on the front and two on the rear of the robot chassis). It has two differential drive motors, each with shaft encoders to determine the approximate position of the robot.

The IntelliBrain Bot from RidgeSoft LLC (2009) are considerably smaller and less sophisticated than the AmigoBot. They are equipped with two IR sensor (forward left and right) and one forward looking sonar and use differential drive for mobility. Two Intellibrain Bots act as the invaders. There are two available behaviors for invaders during the demonstration. First, the robot can operate in accidental mode in which it wanders about randomly. Operating under the second mode, the robot actively seeks out to invade the restricted area.

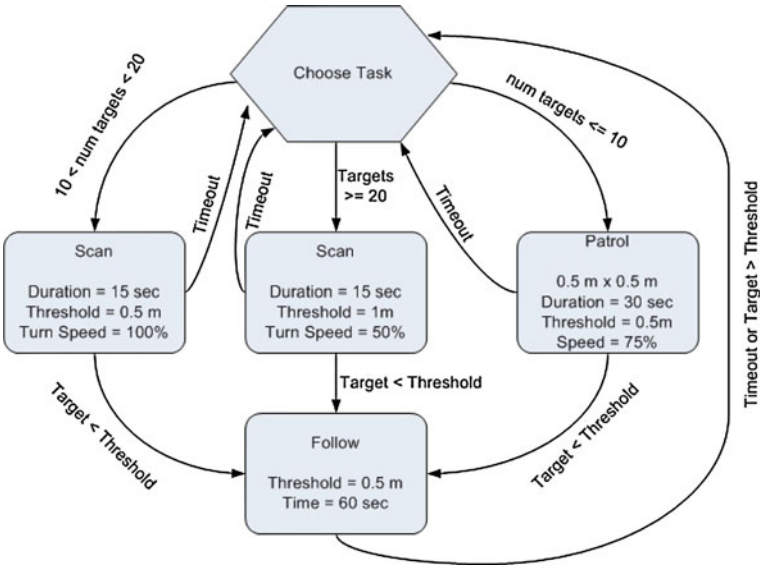

Fig. 5 State-based decision maker presented as a finite state machine

The total time in which one or more invader robots is within the restricted area is measured (i.e. timer starts when one enters and timer stops when the last one leaves). The experiment runs for 5 min and the total time in which the secure area was penetrated is called the *Invasion Time*.

For comparison, a state-based task selector is available. It utilizes the similar expert knowledge, but instead defines states in which the robot may be and the appropriate task and task configuration to handle those tasks. The state-based model is provided in Fig. 5.

6.3 Results and discussion

For the experiment, five scenarios were tested: sentry versus one accidental intruder, sentry versus two accidental intruders, sentry versus one deliberate intruder, sentry versus two deliberate intruders, and sentry versus one deliberate intruder and one accidental intruder. For each scenario, 10 experiments were run using the state-based decision maker and 10 experiments were run with the constraint-based decision maker. The difference in the averages is noted and the percentage of this difference versus the state-based average time is computed to determine the relative improvement or performance loss when using the constraint-based approach as shown in Table 2.

From the experiments, the results showed that the constraint-based approach is better capable of keeping out the invading robots than the state-based approach for all instances except for the condition of two accidentally invading robots. Under the "two accidental" scenario, the likely cause is that the robots may wander right on the edge or corners of the secure area such that they are within the area, but not being adequately detected by the sentry robot. This does not occur when deliberate robots are operating because they are actively seeking the center and very rarely spend time at the edges and corners of the secure zone. For all other cases, there is a significant improvement in performance.

Table 2 Experimental results of sentry robot demonstration

Configuration	Average invasion time (s)		Difference (s)	Percent improvement
	State-based	Constraint-based		
One accidental	75.18	64.33	10.85	14.43
One deliberate	113.00	90.85	22.14	19.60
Two accidental	116.84	129.74	−12.90	−11.04
Two deliberate	256.48	230.53	25.95	10.12
One accidental and one deliberate	140.38	128.86	11.51	8.20

7 Conclusion

The goal of this research was to reconsider constraint satisfaction problems within the domain of robotics. Due to their computational complexity, it has been assumed that CSPs were too computationally intense for real-time decision making necessary for robot operation. This research shows that by adequately framing the problem with a sufficient level of abstract, constraint models can be solved with sufficient timeliness.

The framework builds upon research in constraint programming. Task models were comprised of input variables, output variables, and hidden variables. Constraints defined whether or not the task and its configuration are valid. A constraint optimization algorithm then generates the solution with the greatest or near-greatest utility (given time available).

The first demonstrations used simulation. A custom simulation was constructed to simulate a polar mobile robot surveying an area using remote sensing tools in Antarctica. The robot must choose whether to sit and recharge, or perform a survey with one of its available sensors. The goal of the simulation was to show that using constraints it would be possible to extend the life of the robot further into the year than using a rule-based system. Both constraint and rule-based systems embed expert knowledge regarding vehicle operations from previous field operations.

Early feedback called into question whether or not these tools could be support real-time application on a physical robot. A demonstration was constructed and performed using three robots: one sentry robot using the constraint framework and two invader robots using simple reactive controls. Constraint models were constructed for several common robot tasks including: patrolling, guarding, scanning, and chasing. Using the constraint optimization algorithms, the constraints limited the set of possible choices and the optimizer selected the best available. Using similar decision logic, a state-based controller was less capable of defending the restricted area than the constraint-based controller.

From experiments, it was shown that using constraint satisfaction problems to model the robot task selection is a valid approach. It also shows that the framework produced is generalizable to a number of different problems. While the results were not overwhelmingly positive, statistical analysis shows that the mean performance under each scenario was marked a statistically significant performance increase for most cases. This supports the authors' claim that the application of constraints in robotics warrants further research.

This research is not without its limitations. The design and implementation of the task models is ad hoc without any formal methods for definition, validation, or verification requiring a generate-and-test approach to model development. The use of linguistic variables limits the precision of the models. For some situations such as when a mathematical relationship

defines the constraint, a linguistic variable would be less effective. Lastly, this approach has only been evaluated versus rule/state-based decision makers, and the authors have not compared these results versus a variety of other decision making techniques that exist from the AI community. Each of these issues shall be addressed as future work.

Acknowledgments This work was supported by the National Science Foundation (grant \#OPP-0122520), the National Aeronautics and Space Administration (grants \#NAG5-12659 and NAG5-12980), the Kansas Technology Enterprise Corporation, and the University of Kansas. This work was also supported through an internal research grant from Embry-Riddle Aeronautical University.

References

Akers EL, Stansbury RS, Agah A (2006) Long-term survival of polar mobile robots. In: Proceedings of the 4th international conference on computing, communications and control technologies, Orlando, FL, 20–23 July 2006, vol II, pp 329–333

Bartak R (1999) Constraint programming: in pursuit of the holy grail. In: Proceedings of the week of doctoral students (WDS99), Prague, June 1999

Borning A (1981) The programming language aspects of ThingLab, a constraint-oriented simulation laboratory. ACM Trans Program Lang Syst 3(4):353–387

Center for Astrophysical Research in Antarctica (2009) Current south pole weather data, obtained through the internet: http://www.astro.uchicago.edu/cara/southpole.edu/weather2.html, [accessed 21/12/2009]

Chun HW, Wong RYM (2004) CLSS: an intelligent crane lorry scheduling system. Appl Intell 20(2):179–194

Dechter R (1991) Constraint networks. In: Shapiro SC (ed) Encyclopedia of artificial intelligence. Wiley, New York, pp 276–285

Deris S, Omatu S, Ohta H, Samat P (1997) University timetabling by constraint-based reasoning: a case study. J Oper Res Soc 48(12):1178–1190

Georget Y (2009) Koalog constraint solver: fast constraint solving in Java, obtained through the internet: http://www.informaticians.org/fjcp2004/slides/georget/nii041025.pdf [accessed 21/12/2009]

Gogineni S, Prescott G, Braaten D, Allen C (2003) Polar radar for ice sheet measurements. In: Proceedings of the international geoscience and remote sensing symposium, Toulouse, France, 21–25 July, vol 3, pp 1607–1609

IBM (2009) IBM ILOG CP, obtained through the internet: http://www-01.ibm.com/software/integration/optimization/cp1/ [accessed 21/12/2009]

Kumar V (1992) Algorithms for constraint satisfaction problems: a survey. AI Mag 13(1):32–44

Lever J, Streeter A, Ray L (2006) Performance of a solar-powered robot for polar instrument networks. In: Proceedings of the 2006 international conference on robotics and automation, Orlando, FL, 15–19 May 2006, pp 4252–4257

Mackworth AK (1997a) Consistency in networks of relations. Artif Intell 8(1):99–118

Mackworth AK (1997b) Constraint-based design of embedded intelligent systems. Constraints 2:83–86

Mobile Robots Inc (2009) AmigoBot robot for education and collaborative research, obtained through the internet: http://www.activrobots.com/ROBOTS/amigobot.html [accessed 21/12/2009]

Pai DK (1989) Programming parallel distributed control for complex systems. In: IEEE international symposium on intelligent control, Albany, NY, 25–26 Sept 1989, pp 426–432

Pai DK (1991) Least constraint: a framework for the control of complex mechanical systems. In: The American control conference, Boston, MA, 26–28 June 1991, pp 615–621

Ray L, Price A, Streeter A, Denton D, Lever JH (2005) The design of a mobile robot for instrument network deployment in Antarctica. In: Proceedings of the 2005 international conference on robotics and automation, Barcelona, Spain, 18–22 April 2005, pp 2111–2116

RidgeSoft LLC (2009) Intellibrain-Bot educational robot. Obtained through the internet: http://www.ridgesoft.com/intellibrainbot/intellibrainbot.htm [accessed 21/12/2009]

Russel S, Norvig P (2002) Artificial intelligence: a modern approach. 2. Prentice Hall, Upper Saddle River

Stansbury RS (2007) Constraint-based task selection and configuration for autonomous mobile robots, Ph.D. dissertation, University of Kansas, Lawrence, KS, July 2007

Stansbury RS, Agah A (2008) Autonomous mobile robot task selection and configuration using constraint satisfaction. In: Proceedings of the international conference on automation, robotics, and control systems, Orlando, FL, 7–10 July 2008, pp 103–109

Stansbury RS, Akers EL, Harmon HP, Agah A (2004) Survivability, mobility, and functionality of a rover for radars in polar regions. Int J Control Autom Syst 2(3):334–353
Syrjanen M (1998) Production management as a constraint satisfaction problem. J Intell Manuf 9:515–522
Tsang E (1993) Foundations of constraint satisfaction. Academic Press, London
Zhang Y, Mackworth AK (1994) Will the robot do the right thing? In: Proceedings of artificial intelligence, Banff, AB, May 1994, vol 138, pp 255–262
Zhang Y, Mackworth AK (2005) A constraint-based robotic soccer team. Constraints 7:7–28