12-2008

SB-CoRLA: Schema-Based Constructivist Robot Learning Architecture

Yifan Tang
University of Tennessee - Knoxville

Recommended Citation

Tang, Yifan, "SB-CoRLA: Schema-Based Constructivist Robot Learning Architecture. " PhD diss., University of Tennessee, 2008.
https://trace.tennessee.edu/utk_graddiss/528

To the Graduate Council:

I am submitting herewith a dissertation written by Yifan Tang entitled "SB-CoRLA: Schema-Based Constructivist Robot Learning Architecture." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Lynne E. Parker, Major Professor

We have read this dissertation and recommend its acceptance:

Bruce J. MacLennan, Michael W. Berry, Dongjun Lee

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Yifan Tang entitled "SB-CoRLA: Schema-Based Constructivist Robot Learning Architecture." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Lynne E. Parker, Major Professor

We have read this dissertation
and recommend its acceptance:

Bruce J. MacLennan

Michael W. Berry

Dongjun Lee

Accepted for the Council:

Carolyn R. Hodges
Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

SB-CoRLA: Schema-Based Constructivist Robot Learning Architecture

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Yifan Tang
December 2008

Abstract

This dissertation explores schema-based robot learning. I developed SB-CoRLA (**S**chema-**B**ased, **Co**nstructivist **R**obot **L**earning **A**rchitecture) to address the issue of constructivist robot learning in a schema-based robot system. The SB-CoRLA architecture extends the previously developed ASyMTRe (**A**utomated **Sy**nthesis of **M**ulti-team member **T**ask solutions through software **Re**configuration) architecture to enable constructivist learning for multi-robot team tasks. The schema-based ASyMTRe architecture has successfully solved the problem of automatically synthesizing task solutions based on robot capabilities. However, it does not include a learning ability. Nothing is learned from past experience; therefore, each time a new task needs to be assigned to a new team of robots, the search process for a solution starts anew. Furthermore, it is not possible for the robot to develop a new behavior.

The complete SB-CoRLA architecture includes off-line learning and online learning processes. For my dissertation, I implemented a schema chunking process within the framework of SB-CoRLA that involves off-line evolutionary learning of partial solutions (also called "chunks"), and online solution search using learned chunks. The chunks are higher level building blocks than the original schemas. They have similar interfaces to the original schemas, and can be used in an extended version of the ASyMTRe online solution searching process.

SB-CoRLA can include other learning processes such as an online learning process that uses a combination of exploration and a goal-directed feedback evaluation process to develop new behaviors by modifying and extending existing schemas. The online learning process is planned for future work.

The significance of this work is the development of an architecture that enables continuous, constructivist learning by incorporating learning capabilities in a schema-based robot system, thus allowing robot teams to re-use previous task solutions for both existing and new tasks, to build up more abstract schema chunks, as well as to develop new schemas. The schema chunking process can generate solutions in certain situations when the centralized ASyMTRe cannot find solutions in a timely manner. The chunks can be re-used for different applications, hence improving the search efficiency.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A computational system is not truly intelligent unless it has the ability to learn from past experience, and to learn to adapt to new conditions. Implementing learning in a robot system is necessary for the robot system's efficiency, stability, and flexibility. In prior work, F. Tang and Parker [Tang and Parker, 2005a, Tang, 2006] developed the ASyMTRe approach to automatically generate robot team task solutions for coalitions[1] performing multi-robot tasks. The ASyMTRe approach is inspired by the theory of information invariants [Donald, 1995] and schema theory [Arbib, 2003], and finds team task solutions by configuring the schema building blocks on each robot such that the resulting configuration achieves the specified task with the lowest cost possible. Because the challenge of locating a low-cost configuration of schemas across multiple robot team members is an NP-hard search problem [Tang and Parker, 2005b] (which is also true for other task allocation problems), the ASyMTRe search algorithm that finds potential coalitions is based loosely on the findings of Shehory [Shehory and Kraus, 1999], who showed that for non-super-additive domains, better solutions consist of smaller coalition sizes. These concepts are implemented in ASyMTRe through heuristics that direct the search toward smaller team solutions first. Parker and F. Tang showed, through empirical evaluations, that the heuristic-based centralized ASyMTRe search algorithm generates very good solutions quite quickly for several types of applications. A major benefit of this approach is that it enables robots to easily share sensory, computational, and effector capabilities in solving challenging multi-robot tasks.

My research objective is to extend the ASyMTRe architecture to enable constructivist learning in the multi-robot team. Constructivist Learning is inspired by the human child development theory of Piaget [Piaget, 1952]. It is a method for learning new knowledge and skills based upon past experiences; this type of learning is recognized to be a common method used by humans from infancy to adulthood for lifelong learning[2] [Bruner, 1990]. Constructivist learning in this research refers to the adaptation process through learning from experience and interaction with the environment. There are two major aspects of

[1]The coalition search problem is the problem of finding the appropriate combination of single-task robots that collectively perform multi-robot tasks using instantaneous assignment (taxonomized as ST-MR-IA, per [Gerkey and Mataric, 2004]).

[2]This dissertation uses the constructivist learning theory as inspiration. It is not intended to prove the constructivist theory in any way. I am also aware of the controversy in the psychology research field about the correctness of this theory.

adaptation in constructivist learning [Piaget, 1981]: Assimilation and accommodation, defined as follows:

- *Assimilation* conserves forms, or schemes of behaviors. New forms are incorporated into the system as a new construction, or new assembly, of old forms in the system;
- *Accommodation* modifies forms based on environmental stimulation.

The SB-CoRLA architecture realizes assimilation through “chunking”, which includes off-line Evolutionary Learning (EL), schema chunk harvesting, and online solution search via extended centralized ASyMTRe (ECA) using the harvested chunks. SB-CoRLA also suggests accommodation through online, goal-directed, feedback evaluation. The basic building blocks of these learning processes are schemas and chunks. On one hand, the schemas and chunks implemented in SB-CoRLA have a higher abstraction level than the basic schemas implemented by Drescher [Drescher, 1991] or by Chaput [Chaput et al., 2003, Chaput, 2004], and incorporate higher level reasoning and motor skills. On the other hand, these basic building blocks of SB-CoRLA have a lower abstraction level than behaviors. Conceptually, though not constructionally, a behavior normally consists of multiple perceptual schemas, motor schemas, and the coordination among those schemas. In SB-CoRLA, coordination among the schemas happens through matching information types. The perceptual schemas and the motor schemas can be connected with each other in different ways, in order to evoke different behaviors.

Because much of human learning seems to be based on schema building blocks, my intent is to build upon the schema-based abstraction of ASyMTRe to enable constructivist robot learning. I believe that collections of schemas, called “chunks”, analogous to the *Sensori-Computational Systems (SCSs)* of Donald’s information invariants theory [Donald, 1995], could be learned. Most of the chunks present intermediate solutions of the search problem[3]. Ultimately, my research objective is to enable robot teams to learn and build up chunks constructively, in order to store knowledge from previous search processes, to incrementally build up more abstract knowledge, and to improve the efficiency for future online searches.

This dissertation introduces a Schema-Based, Constructivist Robot Learning Architecture called “SB-CoRLA”. The current solution search strategy of ASyMTRe does not construct chunks that would be amenable to this constructivist learning process. Thus, I developed a schema chunking process that has the benefit of facilitating constructivist learning in multi-robot teams. The schema chunking process includes:

1. an alternative off line search strategy - the Evolutionary Learning algorithm (EL) - that searches for highly-fit solutions and partial solutions off line, for a given robot team composition and team task definition;
2. an off line harvesting process that extracts chunks of schemas from the solutions generated by EL; and
3. an online extended ASyMTRe search algorithm that uses previously learned, relevant schema chunks to find solutions for another robot team composition and another team task.

[3]In this paper, “chunk” and “SCS” are used as synonyms.

The chunking process can find a team solution quickly in certain situations when ASyMTRe cannot find a solution in a timely manner[4]. Schema chunks can be re-used for different team tasks, as long as the task definition requires the information types the schema chunks can provide.

In order to evaluate the chunking process, the following simulations and observations are performed:

1. The time consumption is compared between the original ASyMTRe and the chunking process;
2. The development of team solutions is monitored throughout the EL process;
3. The quality of the final solution generated by the chunking process is compared with the solution generated by original ASyMTRe;
4. The computational complexity for the chunking process is analyzed;
5. The generated chunks and team solutions are reviewed manually to determine their validity.

1.1 Term Definitions

In this subsection, I define several key terms that are used in the SB-CoRLA architecture. This subsection can be used as a quick reference guide to terms, if needed. The *Schema* is the basic building block of SB-CoRLA, and the *Information Type* presents the way of connecting the schemas. A *Sensori-Computational System (SCS)* is a set of schemas connected via information types, and the *SCS repository* is the knowledge base of the SB-CoRLA architecture.

- Schema: A *schema* represents a functional capability of the robot. The abstraction level of a schema is lower than the abstraction level of a behavior, and higher than the abstraction level of a basic motor or sensing capability of a robot. The SB-CoRLA architecture utilizes the following categories of schemas: *Perceptual Schemas (PS)*, *Motor Schemas (MS)*, *Communication Schemas (CS)*, *Valuation Schemas (VS)*, and *Learning Schemas (LS)*, defined as follows:
  - PS: Computational capability to process sensory input
  - MS: Action capability
  - CS: Communication capability, mostly with other team members
  - VS: Computational capability to evaluate feedback
  - LS: Computational capability that processes evaluation produced by VS, and enables behavior modification

  Each schema has a unique set of input information types and one single output information type.

[4]Neither chunking, nor ASyMTRe, is consistently better than the other one in the simulations, as shown in the results discussion.

- Information Type: The term *information type* is used to distinguish from *data type* [Tang, 2006]. While data type identifies the classification of the data, information type identifies the semantics of the information. Some examples of information types are the global position of the robot itself, the global position of another robot, the relative position of another robot, the goal position, different sensory data, and motor control commands.

- Environmental Sensor (ES): *ESs* reflect the sensing capabilities of a robot. Examples of ES are laser, sonar, camera, GPS, bumper, etc.

- Chunk: Same as *Sensori-Computational System.* See below.

- Sensori-Computational System (SCS): An SCS, also called a "chunk", is a set of schemas connected through matching information types. There are three primary levels of chunks:

  - A *first-level chunk* is a set of interconnected schemas that collectively provide one specific information type.
  - A *second-level chunk* is a set of interconnected schemas that collectively provide all necessary information types for one specific type of robot to accomplish one specific task.
  - A *higher-level chunk* is a set of interconnected schemas that represents a partial solution or a complete solution for a robot team task search problem.

- SCS repository: The *SCS repository* is the knowledge base that stores chunks and schemas. In an extended version of the SB-CoRLA architecture, there can be two kinds of SCS repositories: the general SCS repository and the specific SCS repository.

  - The *general SCS repository* stores all known chunks and basic schemas.
  - The *specific SCS repository* stores chunks and basic schemas that are applicable to a specific robot team configuration.

- Constructivist Learning: *Constructivist learning* is a method of learning new knowledge and skills based on past experiences. It has two important aspects, assimilation and accommodation, defined in SB-CoRLA as follows:

  - *Assimilation* reflects the process of incrementally building more complex and more abstract chunks by assembling existing schemas and chunks;
  - *Accommodation* is the process of developing new schemas and chunks by modifying existing schemas and chunks based on feedback derived from interaction with the environment.

- Chunking: The *chunking* process is the assimilation process implemented in SB-CoRLA. It includes three steps:

  - The *off-line Evolutionary Learning process (EL)* learns highly-fit team solutions;
  - The *off-line harvesting process* extracts chunks from the EL solutions;
  - The *online extended ASyMTRe process (ECA)* generates team solutions using previously learned chunks.

1.2 Preview of Results and Contributions

The main contribution of this dissertation is the SB-CoRLA architecture - a schema-based, constructivist learning architecture for robot systems, which includes:

- An off-line evolutionary learning (EL) search algorithm that explores the search space and generates highly-fit partial and complete solutions for various tasks.
- A new randomized ASyMTRe (RA) algorithm that performs the online search process.
- A harvesting technique that extracts chunks from the solutions and partial solutions generated via EL.
- An SCS repository that archives the schemas and the chunks.
- An extended version of ASyMTRe online solution search process - ECA - that utilizes the chunks from the SCS repository to generate online solutions.
- Experimental analysis and comparisons of the chunking process, CA, and RA.
- Exploration of the extension of the SB-CoRLA architecture to include online learning.

Learning is important in a robot system primarily because the robot needs to adapt to unknown environments. The SB-CoRLA architecture enables this adaptation by allowing the robot to find solutions more efficiently based on learned knowledge, and to develop new abilities based on past experiences.

The part of SB-CoRLA implemented in this dissertation differs from other robot learning algorithms in that:

1. it combines off-line schema chunk learning and online solution search using previously learned chunks, while chunks can be re-used not only for the same task where they were learned, but also for different tasks where they can provide the required information types;
2. it is schema-based and utilizes information types to generate information flow within a robot system; and
3. it implements evolutionary learning that naturally selects highly-fit solutions and prunes out solutions with lower relevance.

1.3 Organization of the Dissertation

This dissertation is organized as follows: Chapter 2 presents a review of the related work in schema theory and information invariance, the ASyMTRe architecture, solution searching algorithms, and various SB-CoRLA related learning algorithms. Chapter 3 gives an overview of SB-CoRLA. Chapter 4 describes the assimilation process and Chapter 5 presents the simulation results. Chapter 6 concludes with the main contributions and discusses future work.

Chapter 2

Related Work

Schema theory and information invariants are the inspiration of this research, while ASyMTRe lays the ground work upon which SB-CoRLA is built. This chapter reviews background and related work for schema theory, information invariants, and ASyMTRe. Furthermore, SB-CoRLA enables constructivist learning for more efficient robot team task solution search. This chapter also reviews related works in solution search algorithms, and constructivist learning.

2.1 Schema Theory and Information Invariants

Schemas and information types are the basic elements of SB-CoRLA, inspired by schema theory [Lyons and Arbib, 1989, Arbib, 2003] and information invariants theory [Donald et al., 1994, Donald et al., 1997], respectively. Schemas in SB-CoRLA modify and extend the formal definition of Robot Schema presented in [Lyons and Arbib, 1989] by means of information types and task definitions. Furthermore, SB-CoRLA presents the capabilities of robots at a different abstraction level than the other research efforts presented in this section.

The basic building block of SB-CoRLA is the schema. Several researchers have explored the concept of schemas and developed algorithms using the schema as the basic building block. Lyons and Arbib [Lyons and Arbib, 1989] were the first researchers to incorporate schemas into a programming language for robots. Their language, named Robot Schema (RS), consists of *primitive schemas*, *task schemas*, and *assemblage schemas*. All three schemas have input and output ports. Each port is associated with a data type. Ports are connected via matching data types. There are two kinds of primitive schemas: the *sensory schemas* collect and process sensory data, while the *motor schemas* perform robot motion control. The task schemas contain task definitions about how to connect sensory schemas and motor schemas, and they are also the connecting schemas between sensory and motor schemas. The assemblage schemas are interconnected networks of primitive schemas and task schemas. An example of a very simple assemblage schema consists of one sensory schema connected to one task schema, which is in turn connected to one motor schema. The schemas implemented in SB-CoRLA are similar to the schemas in the RS-model, in that they are connected via matching input and output types. However, schemas in SB-CoRLA are connected via matching *information types*, which adds flexibility and abstraction to the algorithm. There are five different kinds of schemas in SB-CoRLA: Perceptual Schemas (PS), Motor Schemas (MS), Communication Schemas (CS), Learning Schemas (LS), and Valuation Schemas (VS). ES and PS together are comparable to the primitive sensory schema in the RS-model, while MS is similar to the primitive motor schema in the RS-model. There is no task schema in SB-CoRLA. Instead, the task is defined as a set of information types. The idea of an assemblage schema is implemented in SB-CoRLA as *Sensori-Computational Systems (SCSs)*, also called "chunks".

In [Arbib, 2003], Arbib gives an overview of the different theoretical aspects of schemas, and explores schema theory from the neurological perspective. He points out that schemas are recursive in the sense that they can be divided into sub-schemas. To handle the coordination of these sub-schemas, he introduces the notion of Coordinated Control Program (CCP). The CCP coordinates perceptual schemas and motor schemas, and combines several sub-schemas into higher-level schemas. Arbib redefines the neuroscience and cognitive psychological term "working memory" as a short-term memory that holds a range of information relevant for the upcoming actions. The short-term memory is updated in the manner of LRFO (Least Recently accessed First Out). Besides the short-term working memory, the robot can also maintain a long-term memory that stores schemas. This idea has inspired my concept of the general SCS repository in the assimilation process in SB-CoRLA.

Arkin [Arkin, 1987, Arkin, 1998] implemented schema-based behavior control for robots. His approach breaks down behaviors into perceptual schemas and motor schemas. The motor schemas generate motor control outputs. Several motor schemas are combined to produce the desired action. Compared with his approach, the schemas used in SB-CoRLA do not need to have well-defined behavioral functionality, but rather reflect capabilities that can be used in different behavioral implementations. This characteristic adds to the flexibility of SB-CoRLA.

Drescher [Drescher, 1991] and Chaput [Chaput et al., 2003, Chaput, 2004] defined schemas similarly to each other, but differently from the schema definition in SB-CoRLA. Their schemas consist of an action and a set of variables indicating the environmental state before and after the action is taken. No distinction is made between different kinds of schemas. Their research aims to emulate how babies learn to interact with the environment using inherent basic actions, while SB-CoRLA aims for more efficient robot motion control. Not only are the schemas defined differently, they are also at different abstraction levels. The schemas from Drescher and Chaput's systems are more primitive than the ones in SB-CoRLA. Because the schemas in SB-CoRLA are more abstract, the search approach can be more computationally efficient.

The idea of information type is inspired by Donald, et al. [Donald et al., 1994, Donald et al., 1997], who define information invariants as the process of extracting information that is necessary to perform a task from the task definition and/or from the sensors. The goal of their research is to determine what information is required for a robot to accomplish its tasks, or for a team of robots to accomplish their tasks, and how the robot/robots can retrieve this information. While their work is analytical, and concentrates on extracting the information characteristics of a robot task by distributing sensori-computational resources among collaborating robot team members, SB-CoRLA is constructive, and uses the idea of information invariants to achieve automated robot collaboration based on the known information requirements of a robot task.

2.2 ASyMTRe Architecture

F. Tang and Parker [Tang and Parker, 2005a, Tang, 2006, Parker and Tang, 2006] developed the ASyMTRe (**A**utomated **Sy**nthesis of **M**ulti-team member **T**ask solutions through software **Re**configuration) approach to automatically generate robot team task solutions for coalitions performing multi-robot tasks. The ASyMTRe approach was inspired by the theory of information invariants [Donald, 1995] and schema theory [Arbib, 2003].

ASyMTRe addresses the coalition search problem, which is the problem of finding the appropriate combination of single-task robots that collectively perform multi-robot tasks using instantaneous assignment (taxonomized as ST-MR-IA, per [Gerkey and Mataric, 2004]). In the ASyMTRe approach, the search space of this problem consists of basic *schemas* [Arbib, 2003] (ES, PS, MS, and CS), each of which requires and produces certain input(s) and output(s) called *information types*, which can be any kind of sensory and computational data. A task is defined as a set of required information types. The ASyMTRe approach automatically generates task solutions by connecting the schema building blocks via matching information types on each robot such that the resulting configuration completes the specified task with the highest utility[1] possible. Within the context of the schema-based abstraction used to define robot capabilities, the challenge of locating a high-utility configuration of schemas across multiple robot team members is an NP-hard search problem [Tang and Parker, 2005b]. (This is also a general finding for other task allocation problems that use abstractions different from the schema-based abstraction [Gerkey and Mataric, 2004].) The ASyMTRe search algorithm that finds potential coalitions is a heuristic approach based loosely on the findings of Shehory [Shehory and Kraus, 1999], which showed that for non-super-additive[2] domains, better solutions consist of smaller coalition sizes. These concepts are implemented in ASyMTRe through heuristics that direct the search toward smaller team solutions first.

The search space is constrained in a variety of ways. The input to a PS must come from an ES or other PSs. CSs can pass information between PS and/or MS across multiple robots, thus enabling robots to share information between them. The input to an MS comes from a PS or a CS. The control of the robot's motors occurs only through output from MS.

The inputs and outputs of schemas can be interconnected if their information types match. The interconnection process can be illustrated through the following example: Suppose a robot, R, has two schemas, schema S_A and schema S_B. S_A outputs the global position of R, and S_B needs the global position of R as input, in order to compute motor control commands such as a speed and a turning direction. S_A can be connected to S_B via the information type "global position", to enable S_B to produce the information types "speed" and "turning direction". The ASyMTRe process of automatically connecting the schemas through matching information types defines the information flow through a multi-robot system, thus generating the behavior control for a robot coalition. A major benefit of this approach is that it enables robots to easily share sensory, computational, and effector capabilities in solving challenging multi-robot tasks.

[1]The utility of a solution is calculated as a combination of weighted costs and success probabilities of active schemas.

[2]A domain is non-super-additive if the combined effort of separate members does not exceed the sum of the single effort of the same members.

Although ASyMTRe does not include learning, there are many learning possibilities that can be implemented on the foundation of ASyMTRe. SB-CoRLA addresses two learning opportunities: Incremental hierarchical learning (assimilation) and exploratory behavioral learning (accommodation). The assimilation process uses an off-line evolutionary search and learning process to gradually generate and store hierarchical schema building blocks, i.e. to build and reuse sets of interconnected schemas, which can be reused in the online solution search process, in order to increase the solution search efficiency. The accommodation process modifies the existing schemas using feedback evaluation to generate different behaviors. SB-CoRLA uses the same basic building blocks as ASyMTRe. The original schemas and the SCSs in the knowledge base of SB-CoRLA have the same interface as the basic modules in ASyMTRe: they have inputs and outputs in the form of information types, and can be connected with each other via matching information types. In the online solution searching process in SB-CoRLA, the centralized ASyMTRe search algorithm is used to find a solution.

2.3 Solution Search Algorithms

A solution search process evaluates possible solutions based on a problem definition. Some search algorithms aim to find the optimal solution to a problem, while others aim to find a feasible solution. The collection of all possible solutions for one problem is called the "search space" of that specific problem. While brute-force search algorithms search through the entire search space, heuristic-based search algorithms apply some knowledge about the search space, in order to reduce the amount of search, or to find a solution sooner.

Centralized ASyMTRe [Tang and Parker, 2005b] is an anytime search process that aims to find a feasible solution, and to find better solutions given more time. An anytime algorithm produces online results and improves the result quality if there is more time for computation. The ASyMTRe approach employs a heuristic-based search algorithm to find the task solution with the highest utility possible at an early time. Assuming that smaller coalition teams will produce lower cost and higher utility solutions, the heuristic consists of two components, both attempting to preserve as many resources as possible at the beginning of the task assignment process. The first component assigns robots with less available capabilities to the task first. The second component assigns help to robots by first choosing robots with fewer capabilities to provide assistance to a requesting robot. Only if the lower capability robots fail to provide the requested help are higher capability robots chosen. Aside from this heuristic aspect, centralized ASyMTRe uses a brute-force search algorithm that searches through all possible task assignment sequences for the robots. In contrast, SB-CoRLA's assimilation process implements an evolutionary learning search approach to explore alternative search techniques for this NP-hard search problem.

Huang and van de Panne [Huang and van de Panne, 1996] created a decision tree search algorithm to plan for dynamic motions. They reduced the computational complexity of the search process by exploring only promising branches (branches from nodes with high evaluation values) of the decision tree instead of performing an exhaustive search, and by pruning the decision tree and deleting leaves that lead to failed actions to reduce the search space for subsequent search processes. SB-CoRLA uses a similar principle in the assimilation process through an indexing system, which inspects promising SCSs first.

Hansen, et al. [Hansen et al., 1997] explored the anytime heuristic search. They proposed that non-admissible evaluation functions[3] could increase the speed of the search process, in order to find the first solution faster, and eventually to converge to an optimal solution. They pointed out that an anytime algorithm should aim to optimize the search effort, i.e., to optimize the rate of search time to solution improvement. This criterion cannot be applied to the ASyMTRe search algorithm, because the solution improvement is unknown.

More recently, Jones, et al. [Jones et al., 2006] applied the "Play" component from the framework of Skills, Tactics, and Plays (STP) [Bowling et al., 2004] to define tasks for a robot coalition, and applied their approach called TraderBots, to dynamically assign tasks to the robots. A play consists of a set of *roles* that are assigned to different robots, a collection of *actions* for each role to accomplish, and evaluation functions to determine the degree of goal achievement. Each play is assigned a probability of being selected. The selection of a play happens stochastically, which is beneficial in an adversarial environment. Like my approach, this approach can automatically assign tasks to each robot, and find a substitute robot if one robot fails to perform its task. This task assignment occurs at a higher level than in SB-CoRLA. While each robot is assigned a role in [Jones et al., 2006], this kind of role assignment is not defined in SB-CoRLA. In SB-CoRLA, task assignment occurs on the level of information types.

Levner, et al. [Levner et al., 2006] break down a global coordination task into local search problems by identifying the key skills of successful robots in a specific task domain. They solved the local search problems using heuristics and state space pruning to improve the efficiency of the search process. Their approach is task specific, while SB-CoRLA is more general and can be used across tasks and domains. SB-CoRLA starts with the same abstraction level as ASyMTRe, then incrementally builds a knowledge base with more abstract modular components. With the SCS repository, SB-CoRLA aims for non-task specific search results.

Saffiotti, et al. [Saffiotti and Broxvall, 2005, Saffiotti et al., 2008] combine ambient intelligence[4] and autonomous robotics to develop a system called PEIS (Physically Embedded Intelligent Systems) that consists of simple units that can communicate with each other and exchange information. A solution of a task is found by connecting the right units to create the right information flow. Instead of using sophisticated sensing capabilities, a robot achieves its goal by retrieving information about the environment and other objects from communication. Their work is similar to SB-CoRLA in that the units are connected via matching information. They achieve communication among heterogeneous units through cooperative perceptual anchoring [Coradeschi and Saffiotti, 2003, LeBlanc and Saffiotti, 2008], while SB-CoRLA uses matching information types. PEIS employs both centralized and distributed solution configuration [Lundh et al., 2007, Gritti et al., 2007] to search for a task solution. However their search algorithm differs from SB-CoRLA in that it does not enable learning from past experience. Furthermore, although PEIS and SB-CoRLA both address the ST-MR-IA [Gerkey and Mataric, 2004] task allocation problem, PEIS concentrates on enabling one robot to perform a task with help via information from other

[3]A heuristic is admissible if it never overestimates the cost of reaching the goal.

[4]Ambient intelligence refers to a ubiquitous intelligent interface in everyday life, realized by embedded computing and networking technology in everyday objects.

robots/units in the same system, while SB-CoRLA aims for each robot to perform the task. Finally, the search space of PEIS is much smaller.

In [Wang and de Silva, 2008, Wang and de Silva, 2006, Shan and Tan, 2006, Sheng et al., 2006, Huntsberger et al., 2003, Stroupe et al., 2005], multi-robot coordination for specific tasks is explored. Wang, et al. [Wang and de Silva, 2006, Wang and de Silva, 2008] develop a machine learning algorithm that combines reinforcement learning and genetic algorithms to perform multi-robot transportation. Their approach uses a probabilistic arbitrator to select between the output of the reinforcement learning process and the genetic algorithms process. Shan and Tan [Shan and Tan, 2006] develop a partitioning algorithm to assign the appropriate robot to a task in a robot-sensor network that performs target tracking and interception. Sheng, et al. [Sheng et al., 2006] tackle the area exploration task by implementing a distributed bidding model with consideration of the distance among the robots. Huntsberger, et al. [Huntsberger et al., 2003, Stroupe et al., 2005] develop a distributed, behavior-based, multi-robot architecture called "CAMPOUT" to perform tightly coupled planetary surface exploration. All the previous approaches target specific multi-robot tasks. In contrast, SB-CoRLA is a general architecture that enables cross domain robot learning, and is therefore not limited to specific tasks.

2.4 Constructivist Learning

Piaget [Piaget, 1952, Piaget, 1963, Piaget, 1981] laid the groundwork for constructivist learning in child development by identifying incremental, distinct sensori-motor periods of intelligence development. He identified two basic processes of intelligence development: assimilation and accommodation. The assimilation process assembles existing knowledge in new ways to reflect the external reality. The accommodation process consists of modification and adjustment of the existing knowledge, in order to reflect the external reality that cannot be encompassed by the existing knowledge. Similarly, the SB-CoRLA architecture includes both assimilation and accommodation. While this dissertation concentrates on the assimilation process, accommodation will be explored in future work. Brooks and Mataric [Brooks and Mataric, 1993] defined four robot learning categories: parameter/function learning, environment learning, coordination learning, and behavior learning. SB-CoRLA aims to implement environment learning through assimilation, and behavior learning through accommodation.

2.4.1 Assimilation related Learning

Robins and McCallum [Robins and McCallum, 1999] pointed out the importance of the knowledge consolidation process and suggested the use of pseudo-rehearsal (rehearsing pseudo data from the base population), in order to preserve the old knowledge and to prevent catastrophic forgetting, while integrating new knowledge. Their work is an inspiration to the off-line-learning aspect of SB-CoRLA because it is not only important to find a solution online, but also to preserve the learned knowledge off-line.

Coelho and Grupen [Coelho and Grupen, 2000] developed context-dependent controllers[5] to implement biased Q-learning for the "grasp" behavior. Platt, et al. [Platt

[5]A controller selects, coordinates, and sequences different actions.

et al., 2005, Platt et al., 2006] continued their idea of utilizing a local controller instead of a global controller, and implemented *abstract action schemas* [Platt et al., 2005] and *schema structured learning* [Platt et al., 2006]. An action schema is defined as a set of abstract states, abstract actions, an abstract policy/controller, and an abstract transition function. The schema structured approach learns the transition probability between actions and aims for the best policy to select actions, in order to reach a goal state. Although they used the term "action schema", the schema in their system is a generalized representation of a complete robot behavior, e.g., the behavior "localize-reach-grasp", and is completely different than the schema definition used in SB-CoRLA.

Barto, et al. [Barto and Mahadevan, 2003, Konidaris and Barto, 2006b, Konidaris and Barto, 2006a, Şimşek and Barto, 2006] concentrated on the reward system of hierarchical reinforcement learning and the portability of the learned skills. In [Konidaris and Barto, 2006b], the researchers implemented an *agent-space* instead of a *problem-space* to generate portable skills. In [Konidaris and Barto, 2006a], a robot motivational framework was developed to select among different drives based on comparable rewards. In [Şimşek and Barto, 2006], a new reward function was created, *intrinsic reward*, which records the difference of external rewards between consecutive time steps, and reflects the internal state of the robot, in order to promote exploratory behaviors.

Both Grupen and Barto relied on a Markov Decision Process (MDP) as the basic principle of their algorithms because they tried to find the best control sequence of basic behaviors. MDP has not been implemented in SB-CoRLA because it is not the action sequence, but the allocation of robot resources, that is the goal of SB-CoRLA. In the future, when using the SCS repository to find online solutions, it is imaginable to use MDP to select among the available SCSs.

As previously mentioned, Drescher [Drescher, 1991] and Chaput [Chaput et al., 2003, Chaput, 2004] both developed schema-based constructivist learning models to emulate an infant exploring the environment using basic schemas. Their work concentrated on the biological verification of the constructivist point of view using very basic level schemas that reflected the inherent ability of an infant. Unlike their approach, the emphasis of SB-CoRLA lies in automatically generating robot behaviors by employing higher-level schemas, aiming for less computational complexity. The definition of schema is also different. While the schemas in Drescher and Chaput's work consist of *context* and *result items* that reflect the environment states before and after an action is taken, and the specific actions that cause the change of the environment, the schemas in SB-CoRLA are similar to the schemas used in ASyMTRe. The assimilation process in SB-CoRLA is similar to Drescher and Chaput's approach of building more complex schemas from the basic schemas.

More recently, Dogar, et al. [Dogar et al., 2007] use a similar principle of learning, called "affordance", to implement a similar learning process that combines primitive behaviors to generate more complex goal-directed behaviors by observing the environmental state before and after a behavior is taken. They define "affordance" as a relation instance in the form of (effect, (entity, behavior)). The robot interacts with the environment using basic behaviors such as "move forward", "turn right", and "turn left", to collect laser sensor data that reflects the effect of the behaviors on the environment. The algorithm then uses unsupervised clustering to determine different classes of effects and uses feature selection to select relevant features of the sensor reading. In the training phase, relation instances are learned to link current environmental state to desired goal effect through behaviors. These relation instances are then used online to generate a solution to achieve a goal. Unlike SB-CoRLA, their approach does not deal with multiple robots cooperating. The behavior of a robot is determined solely based on the environmental state. Their approach mentioned invariant properties; however, the invariants in their approach apply to the values of the relevant environmental features, while the information invariants in SB-CoRLA apply to the information requirement to accomplish a goal.

The assimilation process is also called a "chunking" process, because it assembles the existing knowledge into a higher hierarchical level of more complex and/or more abstract knowledge. Doumont [Doumont, 2002] pointed out that people process and memorize information in chunks and build chunks recursively to establish a hierarchical information structure. This chunking technique is often used in neural network based temporal sequence learning to increase efficiency and re-usability [Werbos, 1997, Taylor and Taylor, 2000, Konishi and Fujii, 2004]. In SB-CoRLA, chunking is also used to enhance efficiency and re-usability, except at a higher abstraction level, i.e., at the schema level (sensory schema, motor schema, etc.) instead of at the neural level.

2.4.2 Accommodation related Learning

Tedrake, et al. [Tedrake et al., 2005] combined mechanical design and a statistical actor-critic algorithm [Konda and Tsitsiklis, 1999] for a bipedal robot to learn to walk by fine-tuning the control vector based on feedback. In their approach, learning and execution happen simultaneously. The changes caused by each feedback are rather small, so that the basic gait of the robot is never broken. Their learning occurs on a lower level than the schema-based level.

Simonin, et al. [Simonin et al., 2005] used a Bayesian probabilistic model to represent the correlation between the environment and the possible consequences of the actions of a robot. They generated motor control commands based on a learned probability distribution and caused the robot to exhibit different behaviors, such as wall following. Although the sensori-motor space of the robot is reflected in the probabilistic model, they did not differentiate between motor schemas and sensory schemas, but rather treated the sensori-motor behavior as a whole. By separating motor schemas and sensory schemas, SB-CoRLA gains more flexibility.

Other single-task-single-robot learning algorithms include [Cambron and Peters, 2000, Hart et al., 2004, Benjamin et al., 2004]. In [Cambron and Peters, 2000], sensory motor coordination is learned by identifying relevant sensory data through data analysis. In [Hart et al., 2004], task specific success rates for motor control modules are learned via statistical data mining.

Oztop, et al. [Oztop et al., 2004] developed a schema-based computational model to examine the process of behavioral development (developing a new "grasp" behavior based on the existing "reach" behavior) through active, goal-directed exploration. Their model is strongly related to the infant motor development research and implements a neural architecture in each of the schemas. In their work, the schemas are much more complex than the ones implemented in SB-CoRLA, and they are connected through weight vectors that are determined by feedback. They started with an original model of schemas and schema connections that reflected the "reaching" behavior. They then induced changes in the weight vectors that connected the schemas, as well as the schemas themselves, based on the goal definition of what a “grasp” behavior should achieve, by means of population coding based on the feedback, in order to generate a new, “grasp” behavior. The accommodation process in SB-CoRLA planned for future work is similar in that it also learns from feedback, and the changes are also reflected in certain weight vectors. However, SB-CoRLA is different from their approach because the schema structure is different. The schema in SB-CoRLA is the same kind of basic module used in ASyMTRe and is not neuron-based.

Henrich, et al. [Schlechter and Henrich, 2006, Deiterding and Henrich, 2007] research the problem of automatic adaptation using robot sensing data. In [Schlechter and Henrich, 2006], the robot learns threshold values for different behaviors. In [Deiterding and Henrich, 2007], they define four abstraction levels of robot adaptation and perform a series of experiments to determine promising areas for robot adaptation.

Cherubini, et al. [Cherubini et al., 2007] employ a policy gradient learning technique to perform both parameter learning and behavior learning [Brooks and Mataric, 1993]. They define a task as a combination of basic behaviors; their goal is to find the best strategy, which is a composition of basic behaviors, and the best parameter settings for these behaviors, for a given task. They introduce learning to enhance the performance of the solution search process. In the training phase, similarities between different strategies and parameter relevance are learned, which can be used to reduce the online search space.

It is apparent that all behavioral learning algorithms involve feedback evaluation. In SB-CoRLA, it is also intended for the robot to learn new schemas based on online feedback and the consequent evaluation process.

Chapter 3

Approach Overview

This chapter describes the architecture of SB-CoRLA and gives an illustrative example of how this approach would be used.

3.1 Overall SB-CoRLA Architecture

Figure 3.1 shows the entire SB-CoRLA architecture. In this approach, a team of robots are assigned a task. The robot team configuration and task definition are provided. Each robot has a set of environmental sensors (ESs) that display some sensing capabilities. In addition, a robot also has some computational capabilities, some communication capabilities, and some motion capabilities. Some robots may have limited capabilities, e.g., limited sensing capability, limited computational capability, no motion capability, or no communication capability. The robot capabilities and the robot types are provided in a robot team configuration file. A task is defined as a set of required information types. For instance, a box-pushing task can be defined as a set of the following information types: motor control for pushing direction, motor control for alignment with the box, and information from other team members about their status, e.g. whether they have just pushed the box. The task definition is provided in another configuration file.

The capabilities of each robot are encapsuled in different schemas. A parsing process uses the robot team configuration to extract the available schemas for each robot, including perceptual schemas (PSs), motor schemas (MSs), and communication schemas (CSs). In the future, valuation schemas (VSs) and learning schemas (LSs) will be added for the accommodation process, i.e. the online goal-directed feedback-based learning process. The parsing process also extracts robot types from the robot team configuration, and extracts required information types from the task definition. The information extracted by the parsing process can be used in the off-line evolutionary learning process (EL), the online extended ASyMTRe solution search process (ECA), and the online goal-directed feedback-based learning process. It can also be used to identify relevant chunks from the SCS repositories for the ECA process and the accommodation process.

The general SCS repository is the knowledge base of the system. It stores the original schemas and the more complex SCSs, or "chunks". Chunks are generated and utilized in the assimilation process, also called the "chunking" process. Chunking includes off-line learning and off-line chunk harvesting, as well as online team solution search using

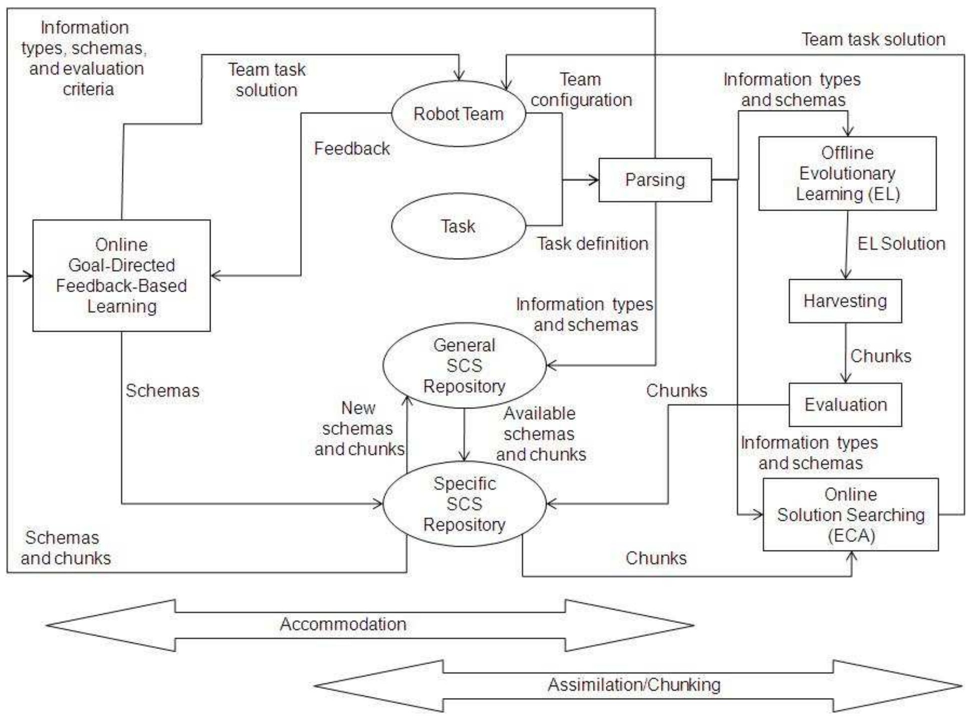

Information types, schemas, and evaluation criteria
Team task solution
Robot Team
Team configuration
Team task solution
Information types and schemas
Feedback
Parsing
Offline Evolutionary Learning (EL)
Task
Task definition
EL Solution
Online Goal-Directed Feedback-Based Learning
Harvesting
Information types and schemas
Chunks
General SCS Repository
Schemas
Chunks
Evaluation
New schemas and chunks
Available schemas and chunks
Information types and schemas
Specific SCS Repository
Online Solution Searching (ECA)
Schemas and chunks
Chunks
Accommodation
Assimilation/Chunking

Figure 3.1: SB-CoRLA Architectural Overview

chunks. For each specific robot team configuration and task definition, based on the present robot types and the required information types, a specific SCS repository is created. A specific SCS repository is a subset of the general SCS repository. Updates in a specific SCS repository are also stored in the general SCS repository periodically. The data in both SCS repositories needs to be kept in sync. One possible way is to create read/write locks for the general SCS repository, so that only one SCS repository can write to the general SCS repository at a time.

The complete SB-CoRLA architecture consists of three processes: Off-line evolutionary learning, online solution searching, and online goal-directed feedback-based learning. These processes are designed for a continuous learning process. They can work together or separately.

- The *off-line evolutionary learning (EL)* process is a part of the assimilation process. It searches for highly-fit[1] robot team solutions for a specific task definition and robot team configuration. Chunks can be extracted from the off-line solutions to be used later.
- The *online solution searching* process generates task solutions for a robot team to perform a current task. While the previously developed *Centralized ASyMTRe (CA)* searches for a team solution using original schemas without considering previously learned knowledge, an extended version of centralized ASyMTRe (ECA) utilizes previously learned chunks that are stored in the SCS repository. Since chunks have a similar interface to the original schemas, they can be used by the online solution searching process in a similar way to the original schemas. The robot team then carries out the task solution with the lowest cost.
- The *online goal-directed feedback-based learning* process collects feedback through interaction with the environment while a robot or a robot team performs exploration. The feedback is then used to modify the existing schemas. This part of SB-CoRLA is not realized in this dissertation, and is part of future work.

3.2 Overview of Assimilation

Figure 3.2 shows the process of assimilation/chunking; this part of the SB-CoRLA architecture is the part of the overall SB-CoRLA architecture that is implemented in this dissertation. In addition to the Centralized ASyMTRe search algorithm (CA) that has been developed in previous work by Parker and F. Tang [Parker and Tang, 2006], I have developed three solution search algorithms: a Randomized ASyMTRe search algorithm (RA), an Evolutionary Learning search algorithm (EL), and an Extended Centralized ASyMTRe search algorithm (ECA). While CA, RA, and ECA are used for online solution search, EL is used for off-line learning.

CA and RA are not part of the chunking process. Their simulation results are used to examine the validity of the chunking process. Both CA and RA generate an exhaustive list of possible solutions (called "potential solutions") via combinations of all available schemas,

[1]The fitness of a solution is calculated with a fitness function. A detailed description of this fitness function is given in Chapter 4, Section 4.1.

then assign each robot in the team one of the solutions based on available schemas on that robot. A robot team solution is a collection of potential solutions, while each robot is assigned one potential solution.

The actual chunking process consists of off-line EL, off-line harvesting, and online ECA. EL does not generate a list of potential solutions. Instead, it generates a graph based on the available schemas in the robot team and required information types. Each graph represents a possible robot team solution. This robot team solution can be incomplete, or partial, i.e. not every robot is guaranteed to be able to provide the required information type. In the off-line EL process, the possible robot team solutions evolve based on a fitness function. When the EL process stops, the best solution is chosen for the next step in the chunking process: harvesting. The harvesting process extracts chunks from the EL solution. The evaluation process assigns priorities to the chunks. These chunks are stored and can be used in ECA. Chapter 4 provides more details about the graph, the chunks, and the chunking process.

3.3 Illustrative Example

To better understand the process of chunking, consider this example: A team of 5 heterogeneous robots needs to navigate from their current positions to a goal position. Each of the robots has a set of sensors and corresponding schemas. Based on the available sensors, each robot is assigned a robot type in advance. Table 3.1 shows the sensors on each robot and their corresponding schemas, as well as the robot types. The communication ability, *"comm"*, is considered a sensor. Each robot also has computational abilities that are defined as schemas *ps2*, *ps3*, and *ps4*. The functionality of each schema is described as follows:

- *ps1*: calculates the robot's self global position;
- *ps2*: provides the goal position;
- *ps3*: calculates global position of another robot;
- *ps4*: calculates the robot's self global position according to the detected relative position and global position of another robot;
- *ps5*: calculates the relative position of another robot;
- *cs*: transfers information between different robots; and
- *ms*: generates motor control signals to move towards the goal.

Each schema has a set of input and output information types. Table 3.2 lists the information types associated with each schema for this example. Some schemas require input information types, such as *ps3* and *ps4*. Other schemas do not require input information type, such as *ps1* and *ps5*. The meaning of each information type is described as follows:

- *f1*: self global position;
- *f2*: another robot's global position;

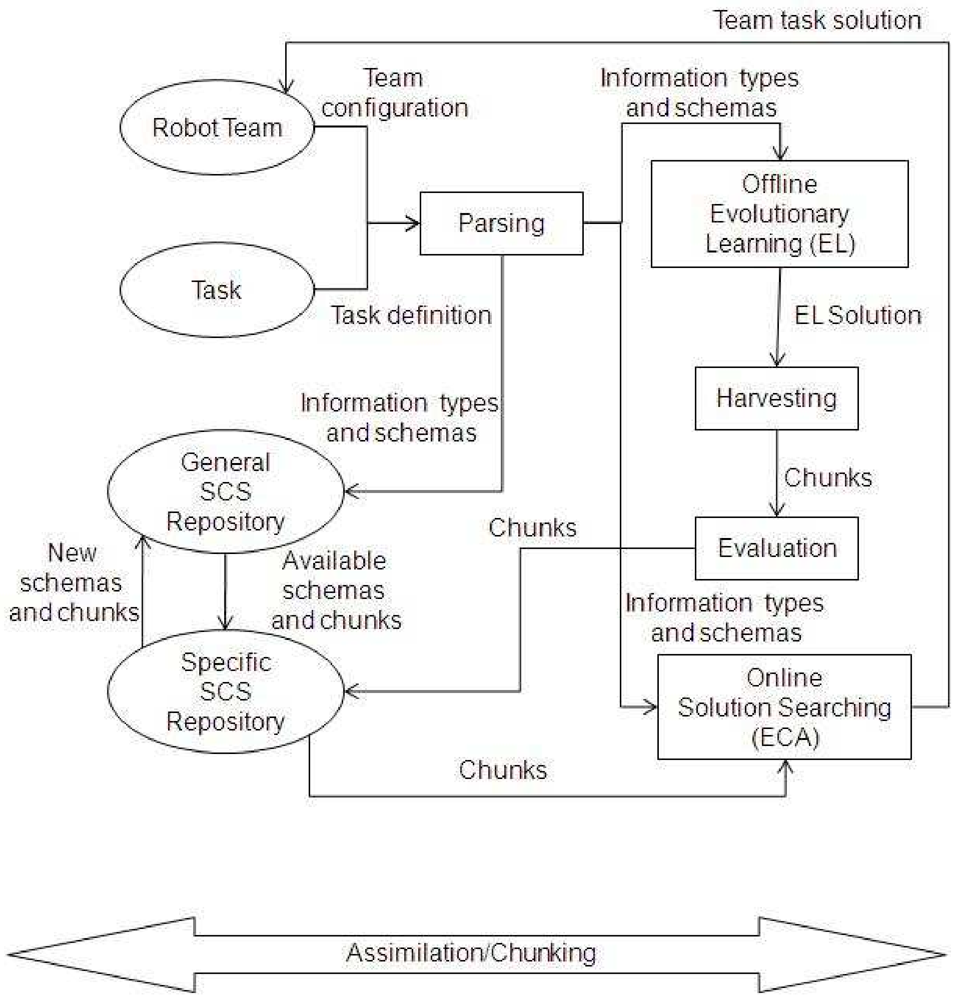


Figure 3.2: SB-CoRLA Architecture Implementation

Table 3.1: Example of 5 heterogeneous robots: their sensors and corresponding schemas

Robot ID	**Robot Type**	**Sensor**	**Schema**
1	6	comm	*cs*
2	6	comm	*cs*
3	5	gps comm	*ps1* *cs*
4	5	gps comm	*ps1* *cs*
5	4	laser camera comm	*ps1* *ps5* *cs*

Table 3.2: Input and output information types of different schemas

Schema	**Input**	**Output**
ps1	none	*f1*
ps2	none	*f4*
ps3	*f1* and *f3*	*f2*
ps4	*f2* and *f3*	*f1*
ps5	none	*f3*
cs	*f1*	*f2*
cs	*f2*	*f1*
ms	*f1* and *f4*	*f5*

- *f3*: another robot's relative position.
- *f4*: goal position
- *f5*: motor control

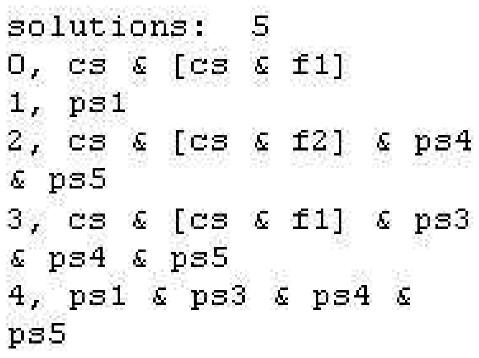
The task in SB-CoRLA is defined as a set of required information types. In this example, the task is defined as one single information type: *f1*. The goal is to find a connection of available schemas within and among the robots, in order for each robot to be able to provide the required information type, *f1*, for motor schema *ms* to reach the goal.

It is assumed that configuration files are available for the robot team configuration and the task definition. The robot team configuration includes robot types, available sensors on each robot, the schemas associated with each sensor, each schema's input and output information types, and the costs of each schema. The task definition contains required information types for the task. A parsing process is applied to these configuration files to extract information for the chunking process.

Recall that both CA and RA generate *potential solutions* and assign a potential solution to each robot. In this example, there are 5 possible ways to connect the schemas, i.e. 5 potential solutions, in order to provide the required information type. Figure 3.3 shows the potential solutions generated by CA/RA for this example. The expression "cs & f1" in square brackets means that the robot gets the information type *f1* from another robot via schema communication. Figure 3.4 shows the graphical illustration of these solutions.

3.4 Overview of Accommodation

The accommodation part of SB-CoRLA is not implemented in this dissertation. In general, a robot team can either carry out a team task strictly according to the task solution generated in the online solution searching process, or perform goal-directed exploration. In the latter case, online feedback can be collected to perform the online goal-directed, feedback-based learning. Figure 3.5 shows a possible online goal-directed, feedback-based learning process. The feedback is processed by the evaluation module, which resides in different Valuation Schemas (VS). The output of the evaluation module induces modifications in the original Learning Schemas (LS) through a learning process. I believe that



```
solutions:  5
0, cs & [cs & f1]
1, ps1
2, cs & [cs & f2] & ps4
& ps5
3, cs & [cs & f1] & ps3
& ps4 & ps5
4, ps1 & ps3 & ps4 &
ps5
```

Figure 3.3: Potential solutions for a team of 5 robots, generated by CA/RA.

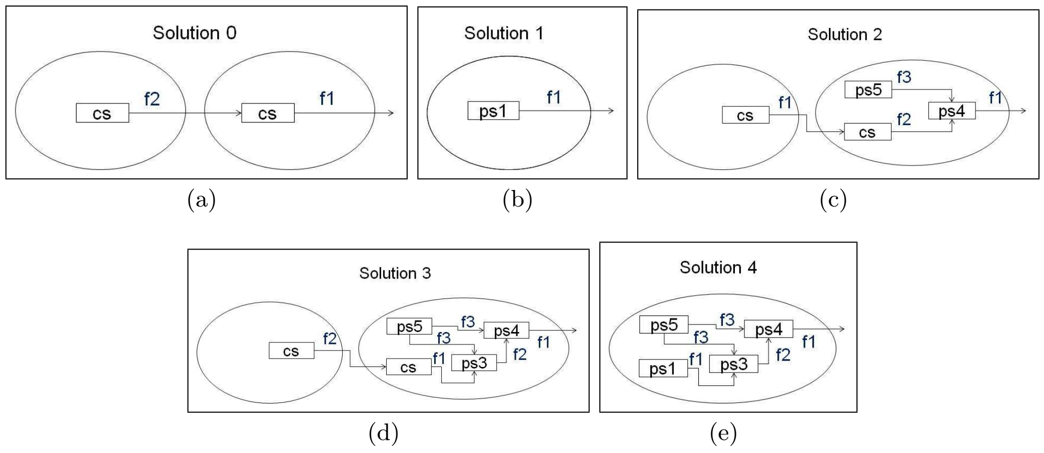


Figure 3.4: Graphical illustration of potential solutions shown in Figure 3.3, for a team of 5 robots, generated by CA/RA.

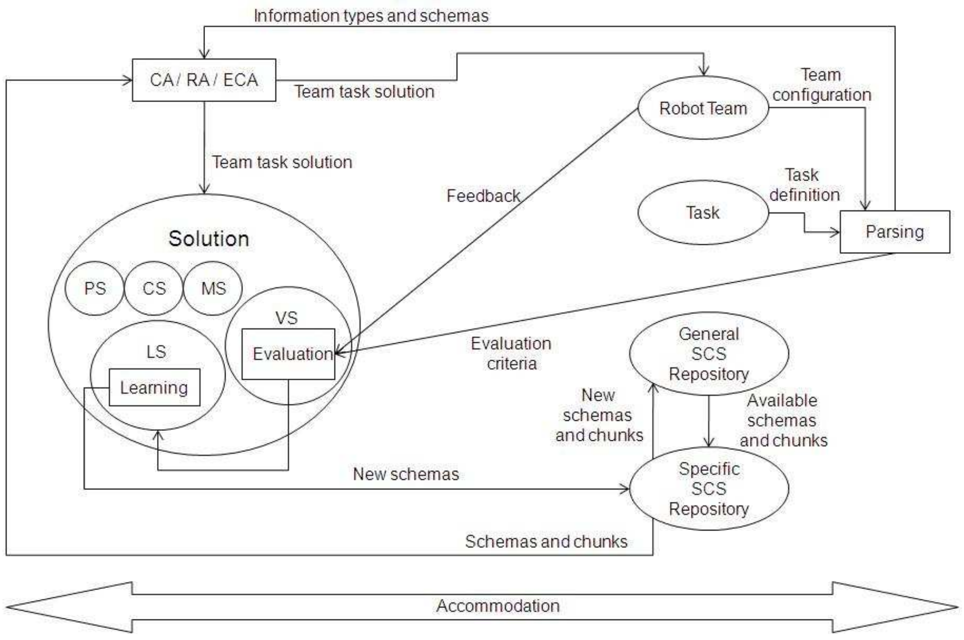


Figure 3.5: Online Goal-Directed Feedback-Based Learning Process

goal-directed exploration can lead to the development of new schemas and new behaviors, and real feedback from task performance can enhance the existing schemas.

Chapter 4

Assimilation

The assimilation process (also called "Chunking") consists of three major functional components:

1. an evolutionary learning search algorithm (EL) to generate off-line robot team solutions;

2. a harvesting process to generate chunks from those solutions; and

3. an online extended ASyMTRe solution search process to use the chunks to generate solutions.

The EL algorithm finds highly-fit partial solutions and complete solutions based on a fitness function. A partial solution is a solution where not all the robots in the team can provide the required information types to accomplish the task. The partial solutions generated by EL are interesting because they contain chunks that are highly-fit. The harvesting process identifies chunks from these solutions. The chunks can then be used in the online solution search process. This chapter describes the assimilation process in detail.

The first objective is to determine if particular search strategies can have the added benefit of facilitating constructivist learning in multi-robot teams. Because much of human learning seems to be based on schema building blocks, the intent is to build upon the schema-based abstraction to enable constructivist learning in multi-robot teams.

Parker and F. Tang [Parker and Tang, 2006] showed, through empirical evaluations, that the heuristic-based centralized ASyMTRe search algorithm generates very good solutions very quickly for several types of applications. However, an open question is whether the ASyMTRe search algorithm for finding coalitions is as good an approximation as possible, given the NP-hard nature of the search problem. The second objective of developing an evolutionary learning search algorithm, therefore, is to explore alternative search techniques for forming coalitions within the schema-based abstraction of robot capabilities. The intent of this objective is to determine whether other search techniques, such as randomized search or evolutionary search, can yield better solutions faster than the heuristic-based search strategy of ASyMTRe for forming multi-robot coalitions.

Since search is a fundamental part of the learning process, the first objective of finding search methods amenable to constructivist learning is closely related with the second objective of investigating alternative coalition search strategies.

To explore alternative search techniques, the centralized version[1] of the previously implemented ASyMTRe search algorithm (CA) is compared with two alternative approaches. The first alternative, labeled “RA” (for Randomized ASyMTRe), makes use of the same fundamental search algorithm of ASyMTRe, but rather than make a greedy heuristic-based search of the potential solutions, it randomly selects possible solutions. The second alternative, labeled “EL” (for Evolutionary Learning), makes use of a genetic algorithm to search the solution space by repeatedly combining highly-fit partial solutions to generate higher utility complete solutions.

The EL algorithm is of particular interest to the constructivist learning objective, since I believe that the highly-fit solutions found in the evolutionary search process can be used to find building blocks for continual learning. These building blocks are also called *chunks*, or *SCSs*. The plan is to enable robot teams to learn and to build up chunks constructively, in order to preserve knowledge from previous search processes, and to improve the search efficiency for future online solution searches. However, I do not want to sacrifice solution quality in making use of the EL search technique in the off-line evolutionary learning process, and need to ensure that the solution quality of the EL technique is comparable to that generated by the CA technique.

The remainder of this chapter is organized as follows. Section 4.1 presents the three search methods: CA, RA, and EL. Section 4.2 outlines the harvesting process and the online solution searching process. Section 4.3 discusses the limitations of the chunking process.

4.1 The Search Algorithms

This section outlines the search approaches used by the three methods — the centralized ASyMTRe approach (CA), the randomized ASyMTRe approach (RA), and the evolutionary learning search approach (EL).

4.1.1 Centralized ASyMTRe (CA) Search Algorithm

As developed by Parker and F. Tang [Parker and Tang, 2006], the centralized ASyMTRe algorithm (CA) is a two-step, anytime algorithm for searching for the proper connections of schemas to accomplish the goal task. The first step is to find all potential schema connections that can provide the required information types for a goal in an individual robot[2]. The second step is to instantiate a specific solution on each robot, by sequentially searching through permutation sequences of individual robots until a simultaneous solution for all robot team members is found. Algorithm 1 shows the detailed steps of CA. A heuristic guiding the search is to find solutions for the less capable robots (i.e., robots that must be part of the solution, but which have fewer schema resources to work with) first, in order to avoid resource shortages. If multiple alternative solutions are found, a utility function is used to select the solution with the highest utility. The CA algorithm is designed to be an anytime algorithm, so that as soon as a valid solution is discovered, it is made available to the robot team. In terms of computational complexity, for n robots

[1]F. Tang and Parker also implemented a distributed version of ASyMTRe [Tang and Parker, 2005c,Tang, 2006]. This dissertation focuses on the centralized version.

[2]These possible connections are called “potential solutions” for the robot in the ASyMTRe approach.

Algorithm 1 The Centralized ASyMTRe (CA) search algorithm. (from Parker and F. Tang [Parker and Tang, 2006])

```
(R, T, U): the robot team configuration, task, and utility
n: the number of robots in the team
m: the number of potential solutions
k: a constant, which specifies the number of iterations

  1. Generate a list of potential solutions of size $m$ by connecting available schemas in
     the entire robot team configuration to satisfy the task's requirements.
  2. For each robot $R_i$, according to its available schemas, generate a list of possible
     solutions for that specific robot. This list can contain at most $m$ solutions, so
     that there is a 1-to-1 match between the potential solutions of the robot team
     configuration and the possible solutions of robot $R_i$.
  3. Sort the list of possible solutions for each robot in descending order of solution
     utility.
  4. Sort the robot team members in ascending order of their available number of
     schemas to generate the first sequential ordering of the robots. [$O(n\log(n))$]
  5. For each robot $R_i$, according to the current ordering: [$O(n)$]
       • For each potential solution $j$ in the sorted list of possible solutions for robot
         $R_i$: [$O(m)$]
           – If $R_i$ can accomplish the task by itself, assign solution $j$ to $R_i$. [$O(1)$]
           – Else check the other $n-1$ robots to see if one can provide the needed in-
             formation. If another robot can provide the needed information, assign
             solution $j$ to $R_i$. [$O(n)$]
       • If no solution is assigned to $R_i$, go to step 7
  6. Calculate the team utility $U$. If $U$ is greater than the utility of the best team
     solution thus far, update the best team solution.
  7. Generate the next sequential ordering of the robots.
  8. Repeat steps 5, 6 and 7 until:
       • All possible sequences of the robot team members have been explored.
       • Or, after $k$ number of trials.
       • Or, after a certain pre-defined period of time.
```

on the team, step 7 of the search process would have to be repeated up to $n!$ times to completely search the solution space (hence, the reason the problem is NP-hard). Thus, the CA approach is a greedy search approach that theoretically searches, in an anytime fashion, all $n!$ permutations of robots, assigning to each robot in order the best solution found. Although Tang and Parker showed through two applications that CA computes the first solution very quickly (i.e., in a matter of seconds), it is unclear whether the solutions found are good approximations to the optimal solution, or whether a solution can be found quickly for any other application as well.

4.1.2 Randomized ASyMTRe (RA) Search Algorithm

The Randomized ASyMTRe algorithm (RA) uses a very similar two-step, anytime search algorithm as used by CA. In doing this, the RA approach first generates potential solutions for each of the robots and then performs a sequential search through each permutation arrangement of robots to assign solutions to individual robots. However, in contrast to the CA approach, the RA approach does not perform a greedy search when assigning solutions to robots. Instead, RA selects viable solutions randomly from among all possible solutions for each robot. Algorithm 2 lays out the details of RA.

4.1.3 Evolutionary Learning (EL) Search Algorithm

The Evolutionary Learning (EL) approach makes use of a genetic algorithm that maintains a population of p individuals, each of which represents a configuration of schemas that may be a possible solution to the robot team coalition task or subtask. Algorithm 3 shows the details of the EL algorithm. Table 4.1 shows the various parameters that must be defined for EL, and their default values[3]. In this section, the concept of EL is explained. Section 6 presents more implementation details for EL.

In the EL approach, an initial population is created that consists of individuals having random connections of schemas, with the following restrictions: first, schemas can only be connected if they have matching information types; and second, connections across different robots can only occur between communications schemas. As the initial population is built, the number of interconnections between schemas on different robots (which I call *inter-robot* connections) and between schemas on the same robot (which I call *intra-robot* connections) are governed by two connection rates specified by the user: the inter-robot connection rate, ρ, and the intra-robot connection rate, κ. Note that these individuals do not necessarily represent complete solutions, since they may not fully (or even partially) solve the task given to the robots. This maintenance of partial solutions during the search process is one of the principal ways in which the EL algorithm differs from the CA and RA algorithms. These partial solutions contain chunks of schemas that solve important subtasks.

As with any genetic algorithm, the fitness value of each individual, F, is determined after each new generation is created through either initialization or evolution[4]:

$$F \quad = \quad w_c \cdot (c/c_{max}) + w_x \cdot x + w_q \cdot (q/q_{max}) + w_u \cdot (u/n)$$

[3]GA related parameter settings are explored in [Goldberg, 2002].

[4]See Table 4.1 and Algorithm 3 for parameter definitions.

Algorithm 2 RA: The Randomized ASyMTRe search algorithm.

(R, T, U): the robot team configuration, task, and utility
n: the number of robots in the team
m: the number of potential solutions
k: a constant, which specifies the number of iterations

1. Generate a list of potential solutions of size m by connecting schemas to satisfy the task's requirements.
2. For each robot R_i, according to its available schemas, generate a list of possible solutions.
3. Generate the next sequential ordering of the robots.
4. For each robot R_i, according to the current ordering: $[O(n)]$
    - For each **randomly selected** solution j in the list of potential solutions of robot R_i: $[O(m)]$
        - If R_i can accomplish the task by itself, assign solution j to R_i. $[O(1)]$
        - Else check the other $n-1$ robots to see if one can provide the needed information. If another robot can provide the needed information, assign solution j to R_i. $[O(n)]$
    - If no solution is assigned to R_i, go to step 6
5. Calculate the team utility U. If U is greater than the utility of the best team solution thus far, update the best team solution.
6. Repeat steps 3, 4, and 5 until:
    - All possible sequences of the robot team members haven been explored.
    - Or, after k number of trials.
    - Or, after certain pre-defined period of time.

Algorithm 3 The Evolutionary Learning (EL) search algorithm (see Table 4.1 for definition of EL parameters).

```
(R, T, F): robot team configuration, task, and fitness
n: number of robots in the team
m: number of different kinds of schemas in the system
$g_{max}$: max. number of generations
c: aggregated cost of active schemas
x: complexity of an individual solution
q: number of required information types
u: number of robots that can achieve their individual goals
$c_{max}, q_{max}, u_{max}$: max. values for c, q, and u
  1. Generate a list of upto $nm$ available schemas based on $R$.
  2. Initialize the first population of size $p$ by connecting the schemas while respecting
     the following rules: $[O((nm^2 + n^2)p)]$
       • For each robot $R_i \in R$, randomly at rate $\kappa$, connect schema $S_a$ to $S_b$ if $S_a$'s
         output information type matches $S_b$'s input information type.
       • Between $R_i$ and each robot $R_j \in R$ ($i \neq j$), randomly at rate $\rho$, connect CS
         schema $CS_i$ to $CS_j$ if $CS_i$'s output information type matches $CS_j$'s input
         information type.
  3. Calculate $F$ for each individual $p$ using the following formula: $[O(n^2m^2p)]$
         $F = w_c \cdot (c/c_{max}) + w_x \cdot x + w_q \cdot (q/q_{max}) + w_u \cdot (u/n)$
  4. Repeat for $g_{max}$ generations
       • Select $\xi$ individuals using fitness-proportionate selection or tournament se-
         lection for reproduction. $[O(n)]$
       • Randomly at rate $\gamma$, perform pairwise crossover on $p$. $[O(n^2m^2p)]$
       • Randomly at rate $\delta$, perform single point mutation on $p$. $[O(nmp)]$
       • Prune each child individual, which is also a partial solution or a complete
         solution for $R$. This process includes eliminating invalid and redundant con-
         nections, as well as calculating the fitness values. $[O(n^2m^2p)]$
       • Record the best solution if its fitness value is better than the best solution
         thus far.
       • Stop if:
           – Every robot can fulfill its individual goal;
           – Or, $l_{max}$ generations have been generated without fitness improvement;
           – Or, a time limit is reached.
```

Table 4.1: EL parameters and their default values

Name	**Description**	**Default**
p	population size	500.0
ξ	number of individuals selected for reproduction	200.0
γ	probability for crossover	0.6
δ	probability for mutation	0.005
κ	intra-robot connection rate	0.8
ρ	inter-robot connection rate	0.8
w_c	weight for the aggregated cost of active schemas; used to calculate fitness	0.2
w_x	weight for the complexity; used to calculate fitness	0.4
w_q	weight for the percentage of information types required by the goal that are fulfilled; used to calculate fitness	0.0
w_u	weight for the percentage of robots that can achieve their goals; used to calculate fitness	0.4

In the framework of SB-CoRLA, F depends not only on the aggregated cost of the active schemas, c, (which is the criterion also used in CA and RA to calculate the cost of the solution), but also on the complexity of the solution, x, and the degree of goal achievement, q and u. The value of x is measured by the total number of schema connections for that solution, and is normalized to the range $[0, 1]$. The degree of goal achievement is measured in two ways: 1) by the percentage of information types that are required by the goal and that are fulfilled (q/q_{max}), and 2) by the percentage of robots that can fulfill their individual goals (u/n). F is calculated as a weighted sum of the normalized values of c, x, q, and u. The weight for each factor is domain-specific and determined by the user.

The evolutionary process for off-line learning in the framework of SB-CoRLA consists of selection[5], single point crossover operations, and single point mutations. In the crossover operation, a crossover point is a randomly selected connection between two random schemas S_i and S_j in one randomly chosen individual solution. In the crossover process, another individual solution is randomly chosen to be the other parent, and then the connection $S_i \rightarrow S_j$ and all the connections that schema S_j are connected to are swapped between the parents. The connections between schemas are uni-directional, indicating the direction of information flow. For example, Figure 4.1 shows one crossover process at the crossover point $S_1 \rightarrow S_5$. Mutation is the process of randomly adding or deleting a connection in an individual solution.

This evolutionary process is repeated over multiple generations until one of the following conditions is fulfilled:

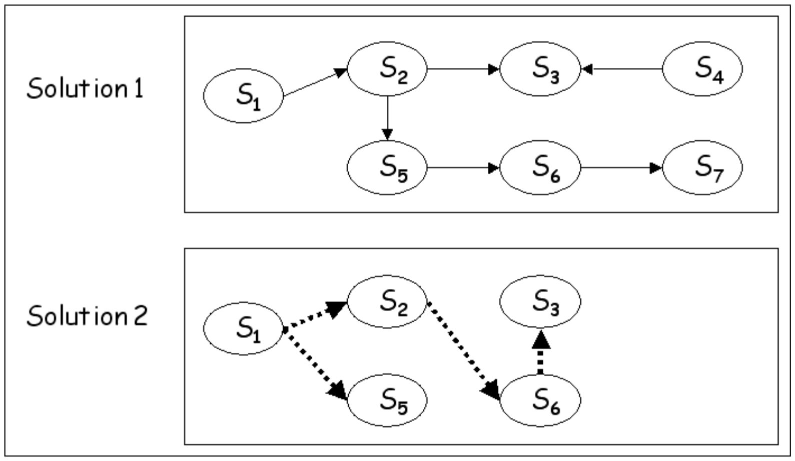
- every robot can fulfill its individual goal;
- the solution quality has not improved in l_{max} generations;
- a time limit is reached; or
- g_{max} generations have been created.

For n robots, m different kinds of schemas, and p individual solutions in each population, there are up to mn available schemas in the search space, and up to $(mn)^2$ possible ways of connecting the schemas. For a maximum of g generations, the EL computational complexity for initializing a population, performing genetic operations, and pruning and evaluating the generated solutions is $O((mn)^2pg)$.

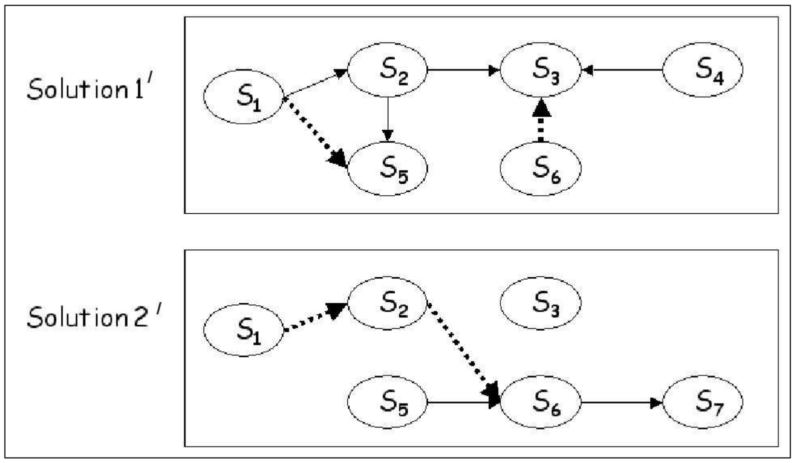
4.1.4 Comparing CA, RA, and EL

Table 4.2 shows a comparison of some of the key characteristics of the three search strategies. Because none of the three algorithms performs a true exhaustive search of the entire search space, no global optimum can be guaranteed with any solution. Compared with CA and RA, EL is more flexible because it can generate partial solutions with progressively improved quality by building upon previous discoveries, as shown from results presented in Chapter 5. This ability of the EL algorithm to identify partial solutions and progressively improve the quality of the solutions without exhaustively searching the entire search space is important for the ultimate constructivist learning objectives. This is distinct from the

[5]The user can choose between Fitness Proportionate Selection and Tournament Selection.



(a)



(b)

Figure 4.1: (a) Parents before crossover: Solution 1 and Solution 2; (b) Children after crossover: Solution 1′ and Solution 2′. The solid arrows indicate the uni-directional connections between schemas in the parent solution Solution 1, and the bold dashed arrows are the connections in Solution 2. The connection between $S1$ and $S5$ is randomly chosen to be the single crossover point. The connections $S_1 \rightarrow S_5$ and $S_6 \rightarrow S_3$ are swapped from Solution 2 to Solution 1 to create the child solution Solution 1′. The connections $S_5 \rightarrow S_6$ and $S_6 \rightarrow S_7$ are swapped from Solution 1 to Solution 2 to create the child solution Solution 2′. Note that because the connections are uni-directional, connection $S_3 \rightarrow S_4$ is not swapped.

CA and RA search processes, which do not have mechanisms for making use of these partial solutions. Instead, the CA algorithm discards these partial solutions (i.e., valid schema connections that lead to the provision of some, but not all, required information types) that have been generated during the search process for each permutation arrangement of robots and starts the search process anew for the next robot sequence. Hence, for the second research objective of constructivist learning, the CA and RA approaches do not appear well-suited. On the other hand, the EL approach is shown to be competitive with the CA and RA approaches, and thus forms a solid foundation upon which to build constructivist learning techniques.

4.2 The Harvesting Process and ECA, the Online Solution Search Process

EL generates partial solutions and complete solutions with high fitness value. These solutions contain schema chunks that are valuable building blocks for robot team solutions. The harvesting process extracts first-level chunks from the solutions generated by EL. Recall that a first-level chunk is a set of interconnected schemas that collectively provides one specific information type. These chunks can be used both in an online search process, and in a future off-line learning process. Algorithm 4 shows the detailed steps of the harvesting process.

The solution generated by EL off-line contains interconnected schemas. This solution can be either a partial solution, i.e., not all the required information types for the task are fulfilled for all the robots in the team; or a complete solution, i.e., all the required information types are fulfilled. To extract first-level chunks from this solution, the harvesting process first determines the fulfilled information types. For each fulfilled information type, the harvesting process backtracks the information flow of that information type, and adds all the involved schemas in one first-level chunk.

After the first-level chunks are extracted, they are then stored in the SCS Repository. Figure 4.2 shows the general format of chunks with an example first-level chunk. The single "$" sign indicates the beginning of a chunk, while the double "$" signs indicate the end of a chunk. The line starting with the single "$" sign contains information about the chunk ID, the costs of all involved schemas in this chunk, the number of active schemas in this chunk, the number of outputs this chunk provides, the type of robot that provides this output(s), and the ID of the robot that provides this output(s). The line starting with the word "robot" contains information about how many robots are involved in this chunk, as well as the type of the robots, and their IDs. The other lines start with a name of a schema, and contain information about which schemas are connected with each other, the robot type and the robot ID these schemas belong to, and whether or not these schemas are from a helper robot. In this example, a first-level chunk consisting of one schema, $ps1$, is shown. The chunk can be used for robot type 5, and outputs the information type $f1$.

Figure 4.3 shows a more complex example first-level chunk with more than one robot involved. The chunk in this example involves 2 robots and 5 active schemas. The robot type 6 receives help via communication from robot type 4, and outputs information type $f1$.

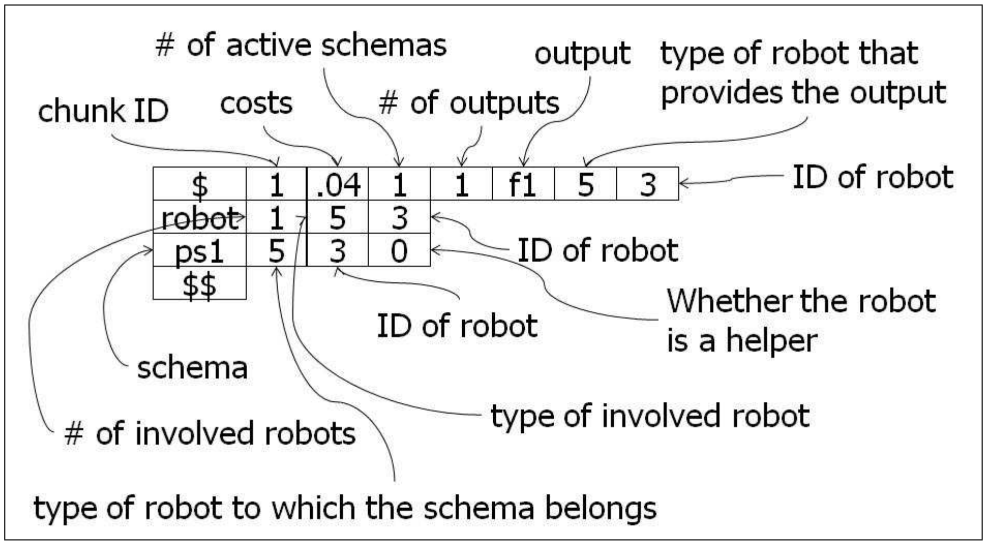
Table 4.2: Comparison between Centralized ASyMTRe search algorithm (*CA*), Randomized ASyMTRe search algorithm (*RA*), and Evolutionary Learning search algorithm (*EL*)

Method	**Computational Complexity**	**Solution Quality**	**Flexible Solution**	**Progressive Improvement**
Centralized ASyMTRe (*CA*)	• Search for one robot sequence: $O(mn^2)$ • Complete search for $n!$ permutation arrangements of robots: $O(mn!)$ (m: number of potential solutions, n: number of robots)	Locally optimal	No	No
Centralized Randomized ASyMTRe (*RA*)	• Search for one robot sequence: $O(mn^2)$ • Complete search for $n!$ permutation arrangements of robots: $O(mn!)$ (m: number of potential solutions, n: number of robots)	Locally optimal	No	No
Off-line Evolutionary Learning (*EL*)	• Search for one generation: $O((mn)^2p)$ • Search for g generations: $O((mn)^2pg)$ (m: number of distinctive schemas, n: number of robots, p: number of individual solutions in the population)	Locally optimal	Yes	Yes

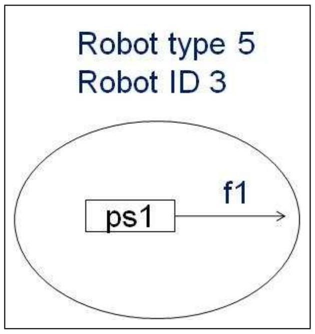
Algorithm 4 The Harvesting Process for First-Level Chunks

```
*Input: EL solution*
*Output: C*
*C: a set of first-level chunks* $\{C_1, C_2, ..., C_i, ..., C_n\}$
$f_{out}$, $f_{in}$: *output/input information types for an arbitrary schema*
$S_i$, $S_j$: *schema that outputs/inputs* $f_{out}/f_{in}$

  1. For each fulfilled information type $f_{out}$ from the EL solution, add this information type to $C_i$
     (a) Let $S_i$ = the schema that outputs $f_{out}$, add $S_i$ to $C_i$
     (b) For each $f_{in}$ that $S_i$ requires
          i. Find $S_j$ that provides $f_{in}$, add $S_j$ to $C_i$
         ii. Let $S_i = S_j$, repeat step 1b, until
               • $S_i$ does not require any $f_{in}$, or
               • There exists no $S_j$ that provides $f_{in}$, in this case add $f_{in}$ to $C_i$
  2. Eliminate identical $C_i$, store unique $C_i$
```

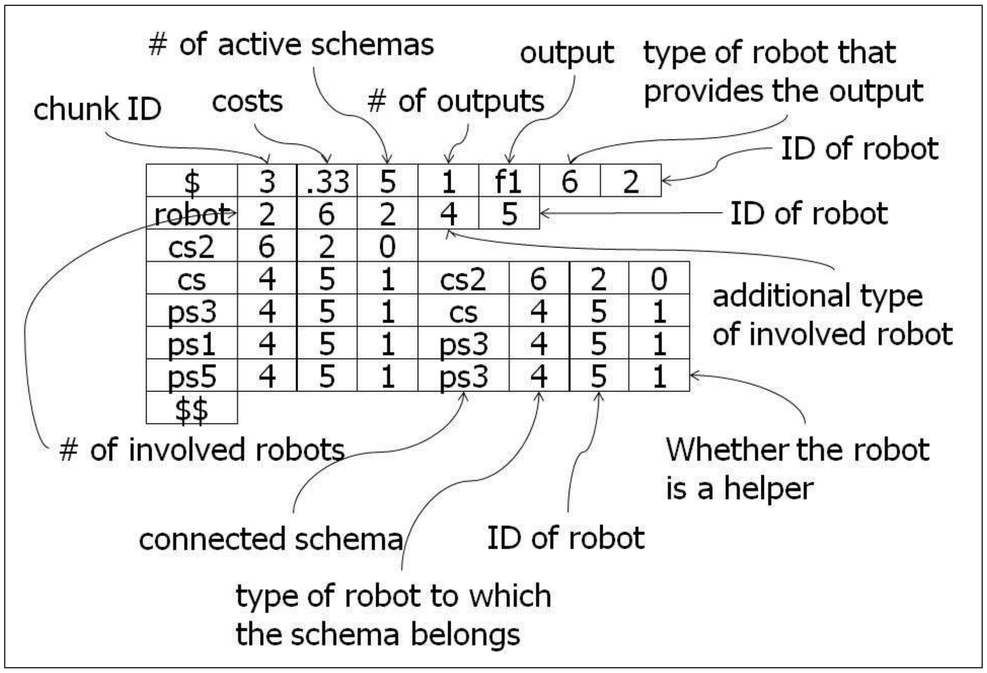


(a)



(b)

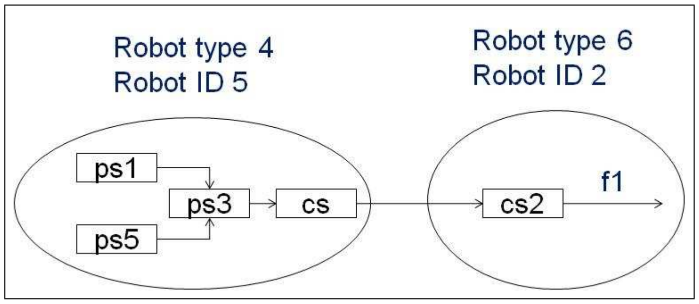
Figure 4.2: An example of a first-level chunk: (a) All first-level chunks are stored in this format (Some chunks are more complex and involve multiple robots, as shown in Figure 4.3.); (b) The graphical display for the first-level chunk shown in part (a).



$	3	.33	5	1	f1	6	2
robot	2	6	2	4	5		
cs2	6	2	0				
cs	4	5	1	cs2	6	2	0
ps3	4	5	1	cs	4	5	1
ps1	4	5	1	ps3	4	5	1
ps5	4	5	1	ps3	4	5	1
$$							

# of active schemas
output
type of robot that provides the output
chunk ID
costs
# of outputs
ID of robot
ID of robot
additional type of involved robot
# of involved robots
Whether the robot is a helper
connected schema
ID of robot
type of robot to which the schema belongs

(a)



(b)

Figure 4.3: Example of a more complex first-level chunk: (a) This chunk involves two robots. Robot type 6 is the robot that outputs the information type $f1$, while robot type 4 helps robot type 6 by providing information through communication schemas. (b) The graphical display for the first-level chunk shown in part (a).

The Extended Centralized ASyMTRe online search process (ECA) utilizes first-level chunks to find robot team solutions. Algorithm 5 shows the detailed steps of the ECA process. At first, ECA uses relevant first-level chunks from the SCS Repository to generate second-level chunks for each robot type in the robot team based on the task definition. A first-level chunk is relevant for the current search process if: (a) its output information type is required in the task definition; and (b) it is for a robot type that exists in the team of robots. To create second-level chunks, the relevant first-level chunks are sorted first according to the robot type they belong to, then according to the information type that they provide. ECA combines the first-level chunks that belong to the same robot type and provide different information types, to generate second-level chunks. Each second-level chunk provides all required information types for one specific robot type. The task team solution is a collection of second-level chunks with active schemas, and connections between the schemas.

The second-level chunks are then sorted in ascending order of costs, number of active schemas, and the number of involved robots. Second-level chunks with lower costs, fewer numbers of active schemas, and fewer numbers of involved robots are considered more desirable. Each robot type has its own sorted list of second-level chunks. ECA then generates a sequential ordering of the robots according to the robot types, and assigns the second-level chunks to the robots in order. The sequential ordering is a permutation of all robot types in the robot team, instead of a permutation of all robot IDs, as implemented in CA. It is possible to implement an *Extended Randomized ASyMTRe online search process (ERA)*. The difference between ECA and ERA is that ECA assigns more desirable second-level chunks first, while ERA assigns them randomly. Figure 4.4 shows an example of three first-level chunks, and the second-level chunk that ECA generates as a combination of these three first-level chunks. Figure 4.5 shows the graphical display of these chunks.

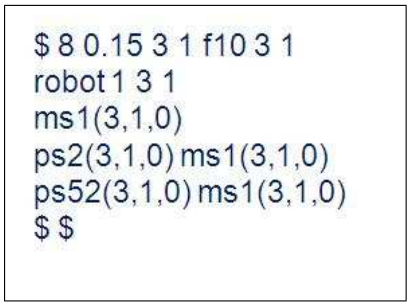
There are several differences between CA (or RA) and ECA (or ERA). Firstly, CA generates potential solutions anew each time it is run. ECA generates second-level chunks based on previously learned first-level chunks. Secondly, CA generates potential solutions for individual robots, which is of computational complexity $O(n)$, where n is the number of robots in the robot team. On the other hand, ECA generates second-level chunks for different robot types, which is of computational complexity $O(m)$, where m is the number of robot types in the robot team, $m \leq n$. Finally, CA generates the sequential ordering of the robots based on their IDs ($O(n!)$), while ECA generates the sequential ordering of the robots based on their types ($O(\frac{n!}{t_1!t_2!...t_i!...t_m!})$, where t_i is the number of robots of robot type i in the robot team configuration).

An evaluation process for the first-level chunks is necessary as the SCS Repository grows, in order to increase the efficiency of ECA (to find relevant first-level chunks, and to generate second-level chunks), as well as to eliminate less desirable chunks. Currently, the first-level chunks are stored in the repository in no specific order. One way to implement an evaluation process could be assigning the first-level chunk index values based on their costs, number of active schemas, number of involved robots, number of connections between active schemas, and other factors. ECA could incrementally retrieve more desirable first-level chunks first, and only retrieve less desirable first-level chunks when necessary. Further development of this idea can be the subject of future work.

Algorithm 5 ECA: The Online Solution Search Process Using First-Level Chunks

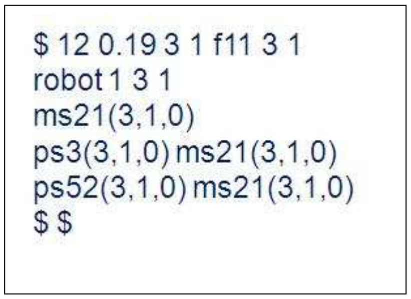
```
Input(R, T, C): robot team configuration, task, first-level chunks
Output(A): robot team solution
$C_f$: first-level chunk
$C_s$: second-level chunk

1. For each robot type $RT_i$ in $R$
     • Find first-level chunks ($C_f$) based on $T$
     • Use all combinations of $C_f$ to generate a set of second-level chunks ($C_s$)
     • Eliminate duplicate $C_s$; as a result, a unique set of $C_s$ remains
     • Sort unique $C_s$ in ascending order of costs of active schemas, number of active
       schemas, and number of involved robots
2. Generate a new robot type sequential ordering
3. For each robot $R_i$ according to the current ordering, assign the best possible $C_s$ to
   $R_i$
     • Update robot ID in $C_s$
     • Find available helper robot if help is necessary
     • Update helper robot ID in $C_s$
4. Record the solution if it is better than the best one generated so far.
5. Repeat steps 2 through 4 until all robot type permutations are visited.
```



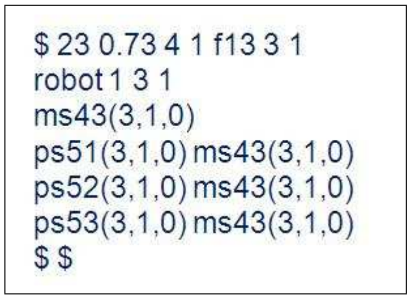
```
$ 8 0.15 3 1 f10 3 1
robot 1 3 1
ms1(3,1,0)
ps2(3,1,0) ms1(3,1,0)
ps52(3,1,0) ms1(3,1,0)
$ $
```

(a)



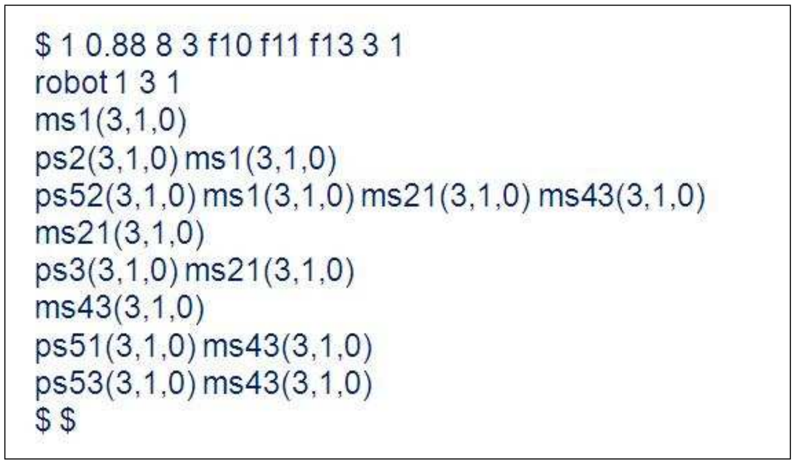
```
$ 12 0.19 3 1 f11 3 1
robot 1 3 1
ms21(3,1,0)
ps3(3,1,0) ms21(3,1,0)
ps52(3,1,0) ms21(3,1,0)
$ $
```

(b)



```
$ 23 0.73 4 1 f13 3 1
robot 1 3 1
ms43(3,1,0)
ps51(3,1,0) ms43(3,1,0)
ps52(3,1,0) ms43(3,1,0)
ps53(3,1,0) ms43(3,1,0)
$ $
```

(c)



```
$ 1 0.88 8 3 f10 f11 f13 3 1
robot 1 3 1
ms1(3,1,0)
ps2(3,1,0) ms1(3,1,0)
ps52(3,1,0) ms1(3,1,0) ms21(3,1,0) ms43(3,1,0)
ms21(3,1,0)
ps3(3,1,0) ms21(3,1,0)
ms43(3,1,0)
ps51(3,1,0) ms43(3,1,0)
ps53(3,1,0) ms43(3,1,0)
$ $
```

(d)

Figure 4.4: Example of three first-level chunks and the second-level chunk that is the combination of these three first-level chunks: (a), (b), and (c) show the first level chunks. (d) shows the second-level chunk that is a combination of these three first-level chunks.

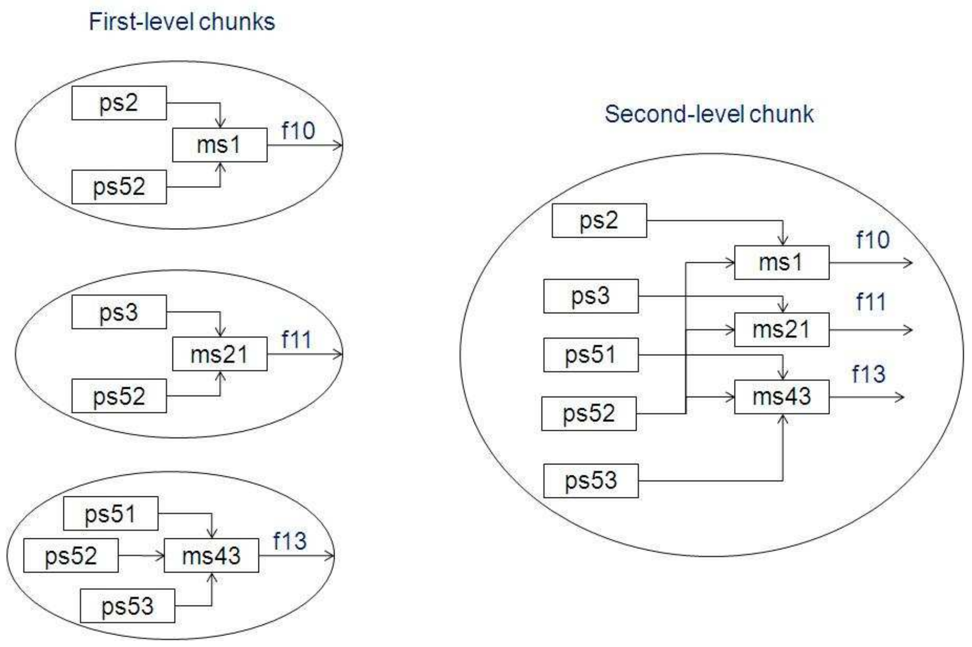


Figure 4.5: The graphical display for the chunks in Figure 4.4: On the left are the three first-level chunks. In the ECA process, those first-level chunks are combined into a second-level chunk, which is shown on the right.

4.3 Limitations of the Chunking Process

The chunking process consists of EL, harvesting, and ECA. All three processes have certain limitations. EL has many parameters that need to be set. While the default parameter settings work well for most of the cases, in some cases, it depends on the expertise of the user to determine the appropriate settings.

The current harvesting process extracts only first-level chunks and combines them to create second-level chunks that fulfill a specific task. One can imagine that directly extracting higher-level chunks might lead to a faster online solution search; however, it will probably require more off-line learning time.

The ECA process assigns chunks to robots, in order to generate a team task solution. It cannot handle new information types and new robot types without an additional off-line learning process. This situation is comparable to a person who learns new skills; this person would also require some time to figure out how to integrate these skills into his/her existing skill set. It is possible to overcome this limitation in three different ways:

1. Implement a hybrid, anytime online solution search process that includes CA, RA, and chunking to search for solutions in parallel. The best available solution from either CA, RA, or the chunking process can be used to assign the current task to the robot team. When given more time, a solution with lower costs can be provided. Figure 4.6 shows an overview of such a hybrid process. Theoretically, including all three online search processes increases the chance of finding a task solution. However, robots often have limited computational capacity. Because the simulation results show that RA outperforms CA in very few cases, it is imaginable to not include RA in the hybrid online solution search process;

2. Include chunks in the CA algorithm. More specifically, use both chunks and original schemas in Algorithm 1, step 1, to generate the list of potential solutions used by CA. Because chunks and schemas have similar interfaces, i.e. they both have input information types and output information types, they can both be used in a similar way to build a potential solution. In order to use a chunk, CA needs to check whether all the schemas involved in the chunk are available in the current robot team configuration[6];

3. Determine similarities between unknown robot types/information types and known robot types/information types, in order to use existing chunks for new, yet similar tasks. This can be done using human knowledge and/or an automatic process that measures the similarities based on different criteria.

[6]Please refer to A.4 for more implementation details.

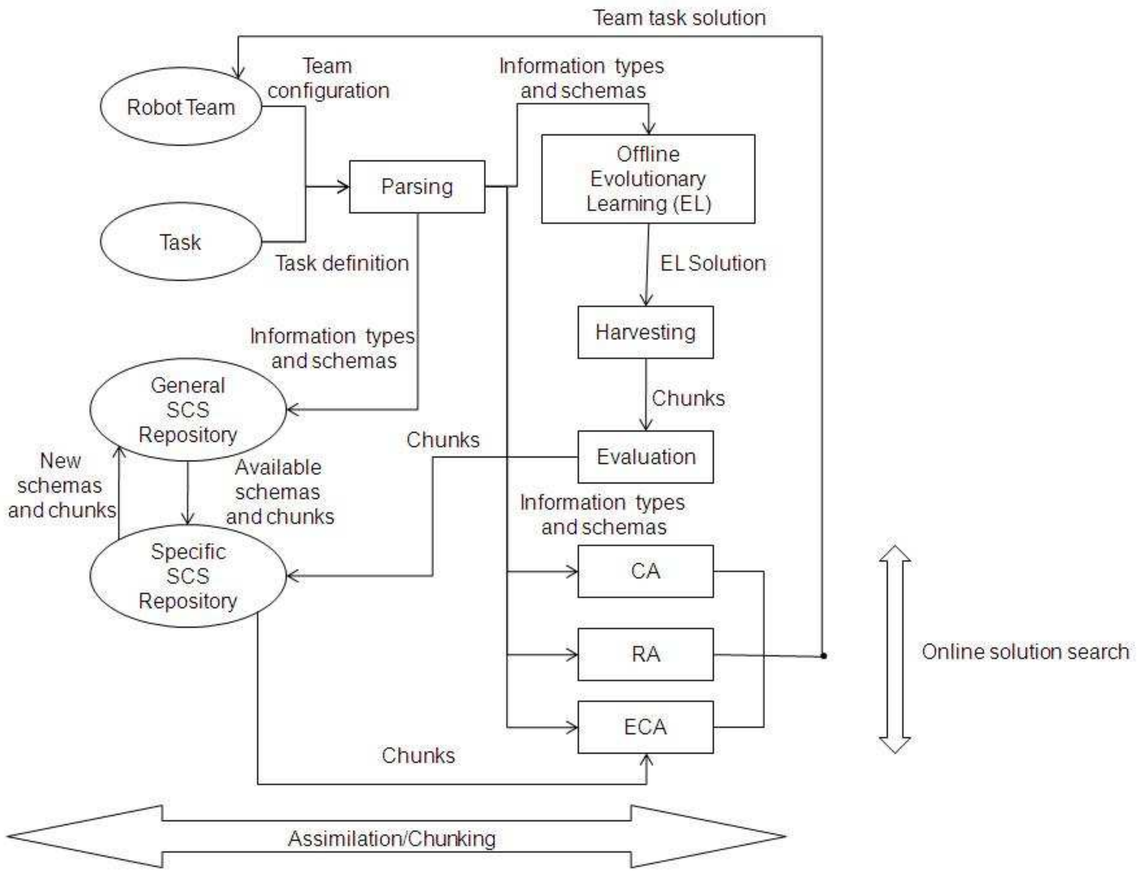


Figure 4.6: This figure shows the combination of CA, RA, and chunking in finding solutions online. All three algorithms are run in parallel. The best available solution from either of these three algorithms within the available time is used to assign the task to the robot team.

Chapter 5

Results and Discussion

This chapter compares the performance of chunking, CA, and RA. In order to do that, I designed four applications and simulated the performance of each of the algorithms on these applications. This chapter examines the following issues:

1. Comparison of the time requirements among CA, RA, and the chunking process, which includes off-line EL, off-line chunk harvesting, and online solution search using schema chunks;

2. Evaluation of the time requirements of the off-line chunking process;

3. Evaluation of the quality of off-line solutions generated by EL, and whether or not EL is able to improve solution qualities over time;

4. Comparison of the solution quality generated by CA, RA, and chunking;

5. Comparison of the differences between chunking and ASyMTRe;

6. Evaluation of the sensitivity of EL for different parameter settings.

The rest of this chapter is organized as follows. Section 5.1 and Section 5.2 describe the simulated applications and the settings used to study the alternative search algorithms, followed by a discussion of the results in Section 5.3. Section 5.4 summarizes the findings from these simulations.

5.1 Description of the Applications

I defined four applications and ran various simulations. The applications are:

- A: multi-robot transportation;

- B: box pushing;

- C: robot formation; and

- D: limited resource.

In these applications, based on the available sensors, each robot possesses different combinations of perceptual schemas. The objective is to find the best team task solution, i.e. to determine which combination of sensors, distributed across which robots, constitutes the highest utility solution for a given task. Figure 5.1 shows examples of implementations of these applications. A detailed description of each application is given next.

Application A requires robots to help each other (through sharing sensory information) in determining their current global position. Various methods of sensor sharing are possible in this application, as implemented by F. Tang and Parker [Tang and Parker, 2005a] on both physical and simulated robots. A robot can have three different sensors: GPS, laser, and camera. Each robot also has communication and computation capabilities. For example, if a robot has a laser, it can use a laser-based perceptual schema to localize itself and calculate its own global position. If a robot has a camera, it can use a camera-based perceptual schema to calculate the relative position of another robot within its sensing range. Using both its own global position and the relative position of another robot, a robot can calculate the other robot's global position, and transmit this information via its communication schema to the other robot.

Application B requires robots to help each other push a box to a goal location. The goal location is indicated with a colored blob. A robot can use its camera to detect its distance to the goal. Again, various methods of sensor sharing are possible in this application, as implemented by F. Tang and Parker [Tang and Parker, 2005a] on both physical and simulated robots. A robot can have three different sensors: laser, camera, and sonar. Each robot also has communication and computation capabilities. In this application, if a robot has a laser, it can apply a laser-based perceptual schema to measure the box's relative position to itself, or activate another laser-based perceptual schema to confirm contact with the box. If a robot has a camera, it can detect the goal location and use a camera-based perceptual schema to calculate its push direction. With sonar, a robot can apply a sonar-based perceptual schema to detect the position of the box.

Applications C and D are designed to test the limitations of CA, and are designed as abstract applications that require certain information flow among the schemas. They are theoretical tests with abstract sensors. Application C has been implemented on physical robots in the SDR project [Parker et al., 2004], however not in the framework of schema-based robot system. Application D has not been implemented on physical robots.

Application C can be thought of as a "robot formation" application, because each robot needs a certain information type, that only one other robot can provide. Intuitively, this would cause the robots to maintain a certain formation. This application consists of n robots. The first robot is the leader of the formation and does not need any information from the other robots. The i^{th} robot needs a unique information type from the $(i-1)^{th}$ robot, indicating the $(i-1)^{th}$ robot's position, so that the i^{th} robot can maintain the formation.

Application D can be thought of as a "limited resource" application, because while a majority of the robot team can use additional information types from a helper robot to increase the utility of its solution, only a few members of the robot team can provide these information types. This application consists of n robots. In this application, except for the last m robots (robots n, n-1, n-2, ..., n-m+1), all robots can accomplish their tasks without help from other robots, i.e., without information communicated by other robots. However, if they can receive external help from other robots, the cost of the solution with

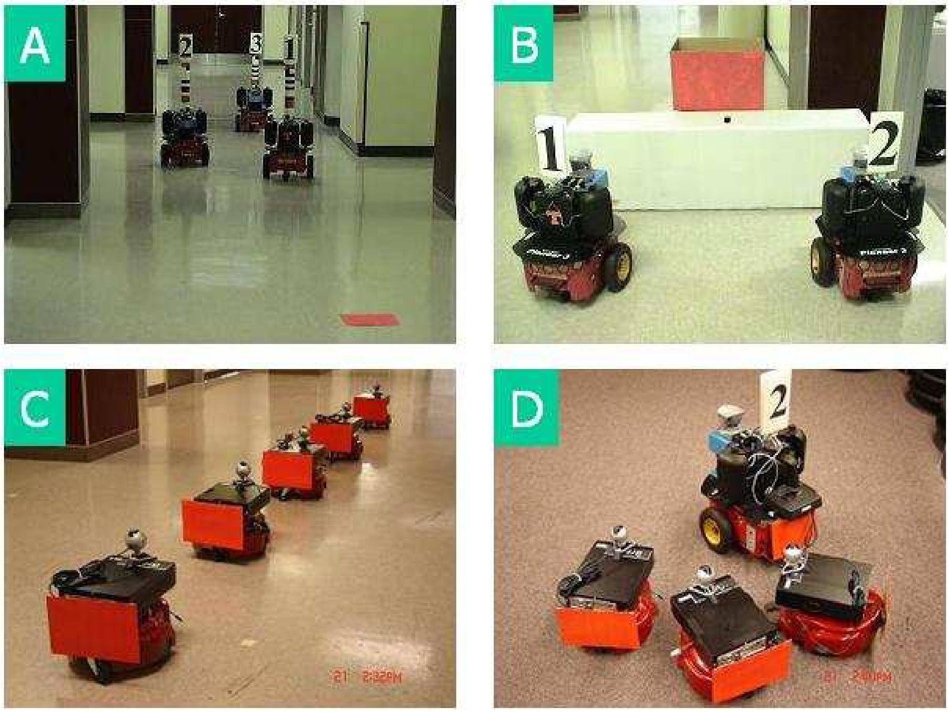


Figure 5.1: Possible implementations of applications A, B, C, and D. Application A and B have been implemented on physical robots by F. Tang and Parker [Tang and Parker, 2005a]. Application C has been implemented on physical robots in the SDR project [Parker et al., 2004] but not in the framework of schema-based robot systems. Application D has not been implemented on physical robots.

Table 5.1: Number of schemas for applications A, B, C, and D

# of robots	App. A	App. B	App. C	App. D
5	40	83	22	43
10	85	166	47	83
15	129	249	72	123
20	182	332	97	163
25	226	415	122	203
50	452	833	147	243
100	904	1660	172	283

external help is lower than the cost of the solution without external help. To constrain the problem, I require that the last m robots must receive external help in order to achieve their goals. However, only the first h robots (robots 1, 2, ..., h) can offer this external help. The available external help in a robot team is sufficient to help the last m robots, but is not enough to fulfill all the requests for external help.

5.2 Experimental Design

The CA and RA algorithms are implemented in C, and chunking is implemented in C++. All the simulations are run on typical present-day Linux PC machines.

The three search strategies CA, RA, and EL, as well as the complete chunking process, including EL, chunk harvesting, and ECA, are tested using heterogeneous robot teams of size ranging from 5 to 25. Occasionally, additional tests are conducted using robot teams of size 50 and 100. Heterogeneous robot teams are composed by randomly choosing different available sensors and consequently different schemas for each robot. Unless explicitly mentioned, the default parameter settings in Table A.3 are used. Wall clock time is measured in the precision of 0.01 second.

While the time requirements vary for each simulation run, the generated solutions are identical for the same parameter settings. Therefore, for the simulations in Section 5.3.1 and 5.3.2 that test the time requirements of the processes, five simulation runs are performed for each application and robot team configuration. The error bars in the figures show the standard deviation of the simulation results. The simulation results presented in Section 5.3.3, 5.3.4, and 5.3.6 that test the solution qualities are from single runs. Table 5.1 shows the number of schemas for each simulation setting that are generated in the parsing process.

Table 5.2: Time breakdown for CA/RA and the chunking (EL + harvesting + ECA) process

Procedure	**CA/RA**	**Chunking**
Pre-processing	Parsing configuration files, generate lists of potential solutions	EL (off-line): Parsing configuration files, initialize the first generation
First online solution	Assign potential solutions to robots	ECA (online): Assign second-level chunks to robots according to robot types
All online solutions	Iterate through permutations of robot IDs	ECA(online): Iterate through permutations of robot types
Off-line learning, one generation	Not applicable	EL (off-line): Evaluate, prune, selection, crossover, mutation
Off-line learning, harvesting chunks	Not applicable	Harvesting (off-line): Extract first-level chunks, combine first-level chunks to generate second-level chunks

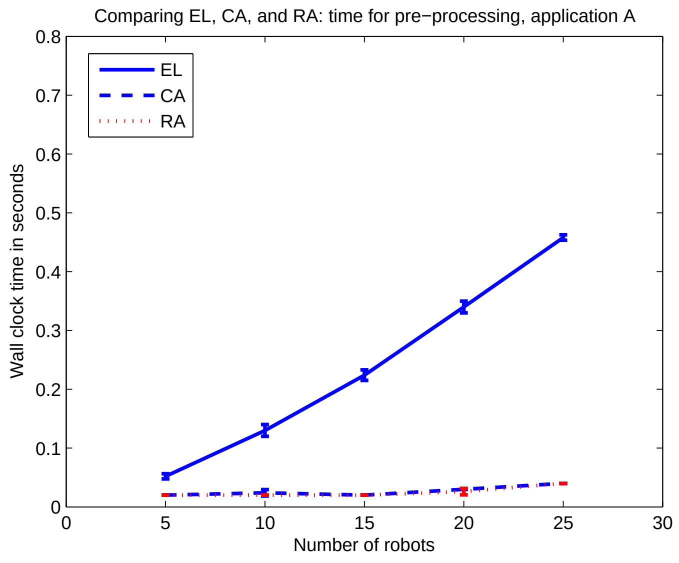
5.3 Results and Discussion

5.3.1 Comparison of the time requirements among CA, RA, and the chunking process

Figures 5.2 to 5.11 show the processing time comparison and simulation results among CA, RA, and the chunking process.

While CA and RA are online solution search processes, chunking is a combination of off-line learning and an online solution search. The processing time can be broken down into 5 different categories as shown in Table 5.2. While chunking, CA, and RA all need pre-processing time, the pre-processing procedure is an off-line procedure for chunking, while it is an online procedure for CA and RA. While chunking uses the pre-processing time in off-line learning to generate EL solutions and finally chunks to be reused in the future, CA and RA require pre-processing for each online solution search. In this section, the EL pre-processing time and the ECA time to generate the first online solution are compared with CA and RA.

Figure 5.2 shows the pre-processing time comparison for application A among EL, CA, and RA. Figure 5.3 shows a closer look of the elapsed time for CA and RA. Figures 5.4, 5.5, and 5.6 show the pre-processing time comparison among EL, CA, and RA, for applications B, C, and D. In Figure 5.6, the elapsed time for CA and RA go off the chart. In order to show the entire graph, Figure 5.7 shows the time comparison for application D in an axis scale different from Figures 5.2, 5.4, and 5.5. EL takes more pre-processing time for application A, B, and C, while CA and RA take more pre-processing time for application



(a)

Figure 5.2: Pre-processing time comparison among EL, CA and RA, for application A. CA and RA are so close that they are indistinguishable on a grey-scale graph. (Each data point in this figure is the average of five runs.)

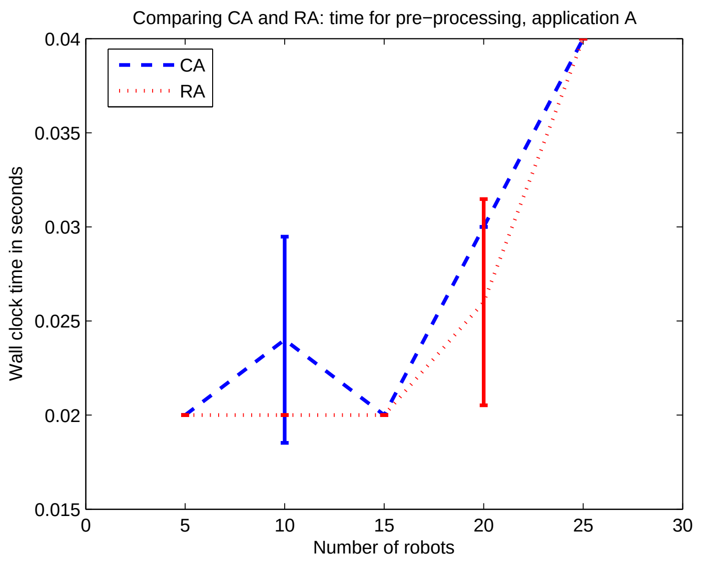
D. While the pre-processing time for CA seems constant for application A, it shows a slowly growing tendency for application B. EL takes the most pre-processing time for application B, because application B has the most available schemas, and thus the most graph nodes for graph generation.

Figures 5.8 to 5.11 show comparisons among ECA, CA, and RA of the time required to find the first online solution for applications A, B, C, and D, respectively. In Figure 5.11, the elapsed time for CA and RA go off the chart. In order to show the entire graph, Figure 5.12 shows the time comparison for application D in an axis scale different than Figures 5.8, 5.9, and 5.10.

The simulation results show that CA takes more time to generate the first online solution for applications A, B and C. Both ECA and RA take almost no time (<0.01 second) to generate the first online solution for application A, B, and C. While ECA takes almost no time to generate the first online solution for application D, Figure 5.12 shows that CA cannot generate an online solution for application D for a team of 15+ robots in a timely manner, and RA cannot generate an online solution for application D for a team of 20+ robots in a timely manner. It takes CA over 2 days to generate the first solution for a team of 15 robots for application D.

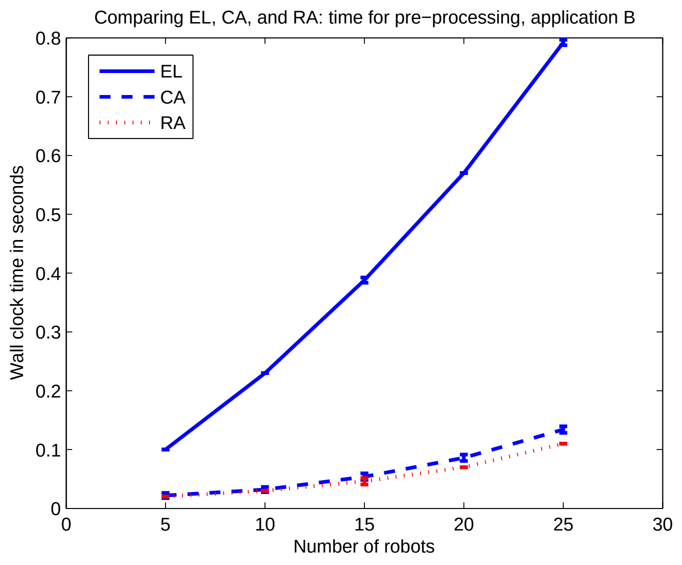
5.3.2 Time requirements of off-line learning in the chunking process

As previously shown in Table 5.2, there are three categories of time requirements for the off-line learning process: EL pre-processing, EL learning, and harvesting. Subsection 5.3.1 has already discussed the EL pre-processing time requirement. This section will discuss



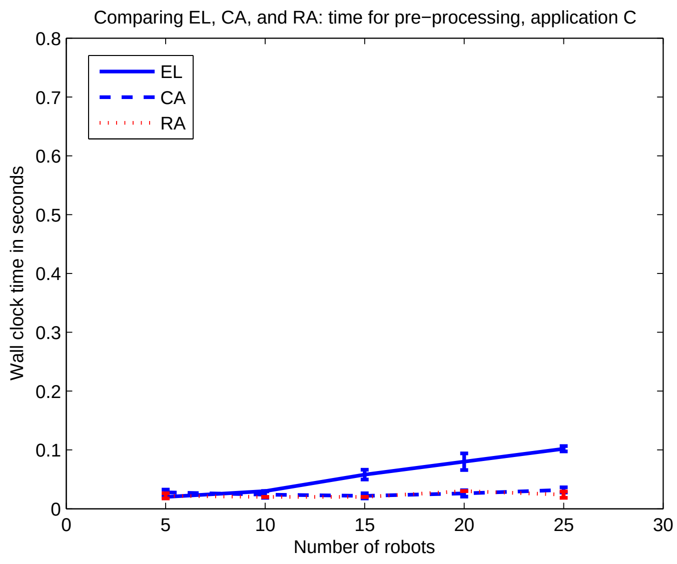
(a)

Figure 5.3: Pre-processing time comparison between CA and RA, for application A. The simulation results only show data variance at two cases, one by CA for a team of 10 robots, and one by RA for a team of 20 robots. Please note that in both cases, the variance is as small as 0.01 second. (Each data point in this figure is the average of five runs.)



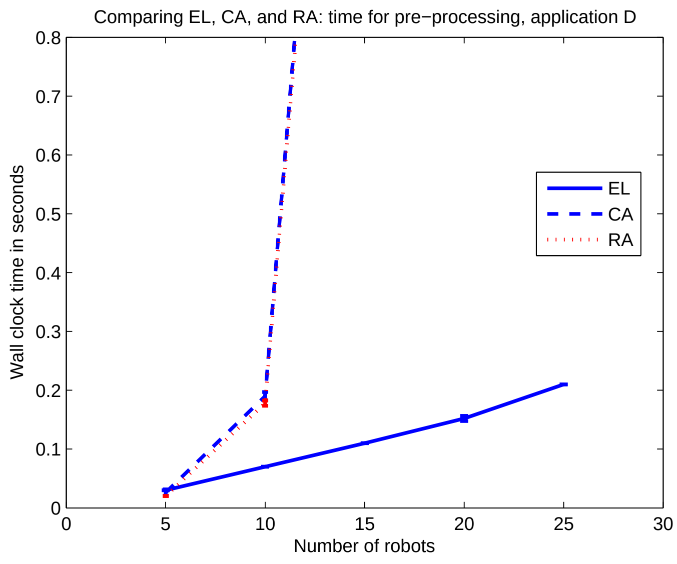
(a)

Figure 5.4: Pre-processing time comparison among EL, CA and RA, for application B. CA and RA are so close that they are almost indistinguishable on a grey-scale graph. (Each data point in this figure is the average of five runs.)



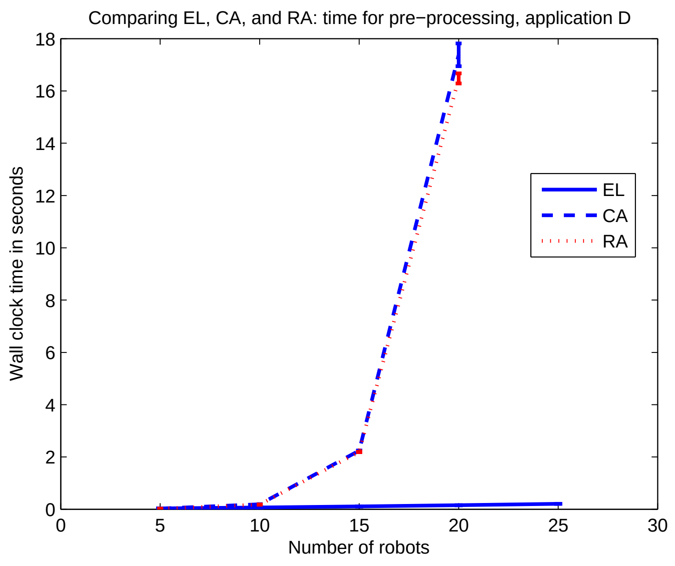
(a)

Figure 5.5: Pre-processing time comparison among EL, CA and RA, for application C. CA and RA are so close that they are indistinguishable on a grey-scale graph. (Each data point in this figure is the average of five runs.)



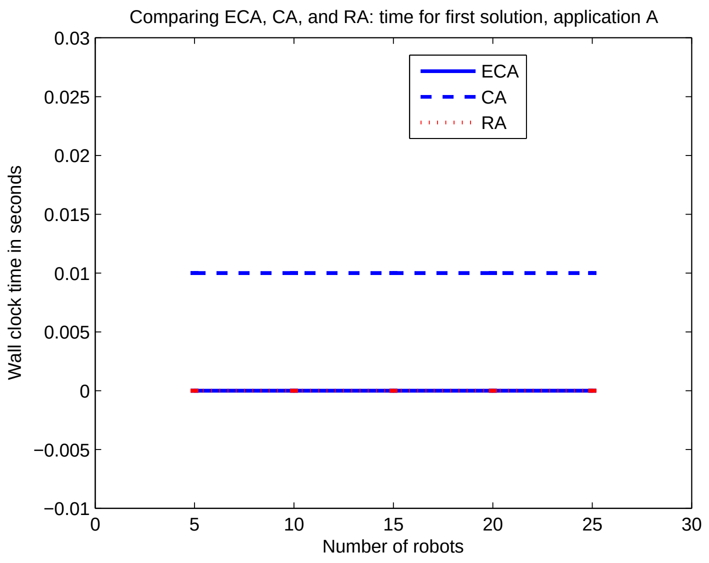
(a)

Figure 5.6: Pre-processing time comparison among EL, CA and RA, for application D. CA and RA are so close that they are indistinguishable on a grey-scale graph. (Each data point in this figure is the average of five runs.)



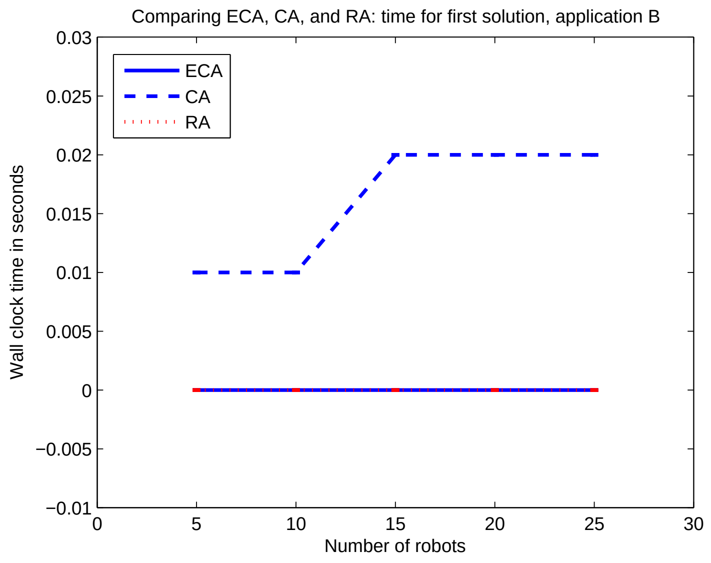
(a)

Figure 5.7: Pre-processing time comparison among EL, CA and RA, for application D, using a different scale than Figures 5.2, 5.4, and 5.5, in order to show the entire graph. CA and RA are so close that they are indistinguishable on a black-white graph. (Each data point in this figure is the average of five runs.)



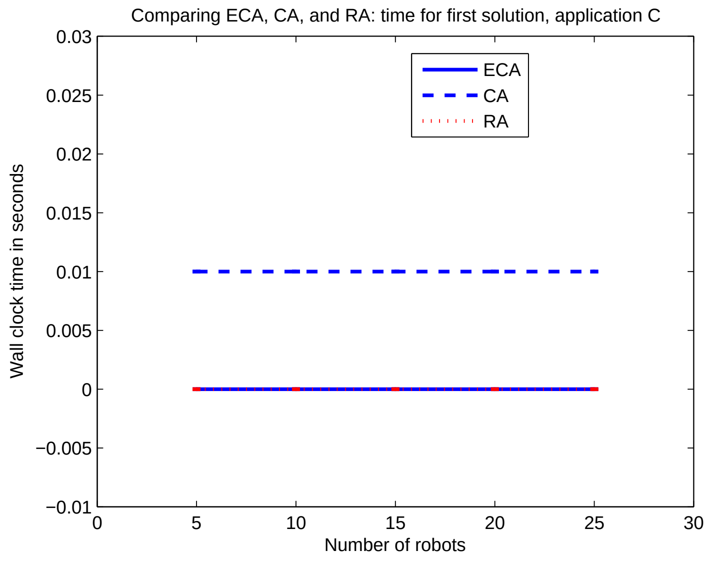
(a)

Figure 5.8: Comparison among ECA, CA, and RA of time required to generate first online solution for application A. The time requirements for ECA and RA are so close that they appear identical on the graph. (Each data point in this figure is the average of five runs.)



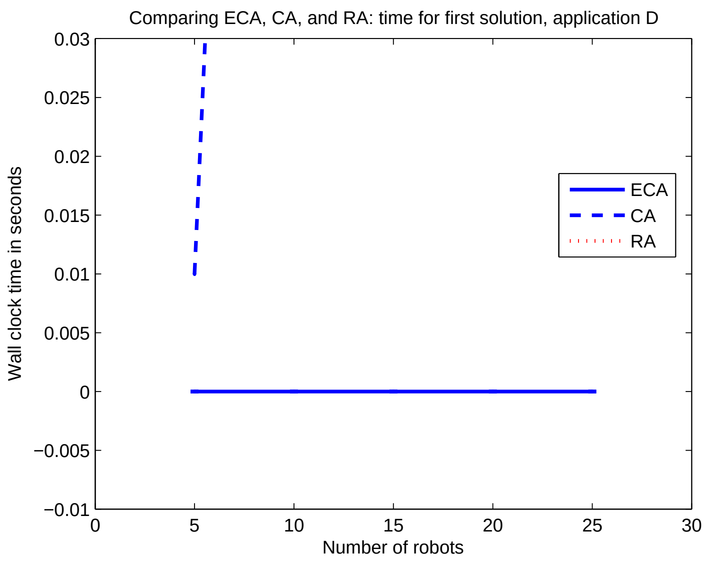
(a)

Figure 5.9: Comparison among ECA, CA, and RA of time required to generate first online solution for application B. The time requirements for ECA and RA are so close that they appear identical on the graph. (Each data point in this figure is the average of five runs.)



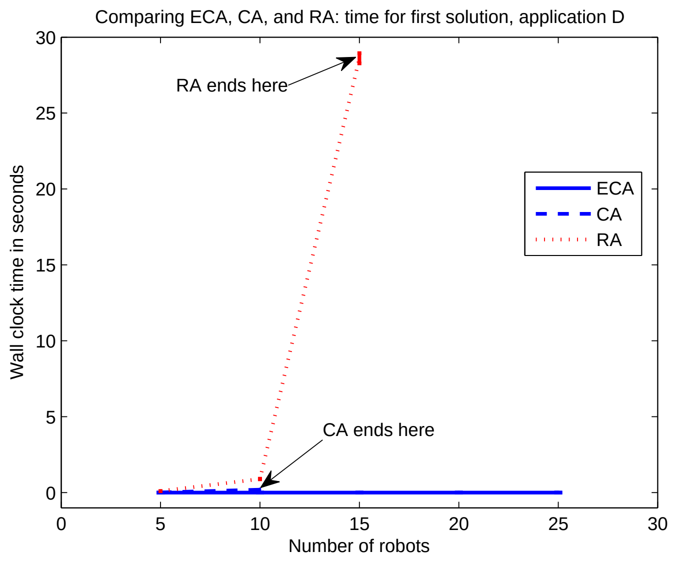
(a)

Figure 5.10: Comparison among ECA, CA, and RA of time required to generate first online solution for application C. The time requirements for ECA and RA are so close that they appear identical on the graph. (Each data point in this figure is the average of five runs.)



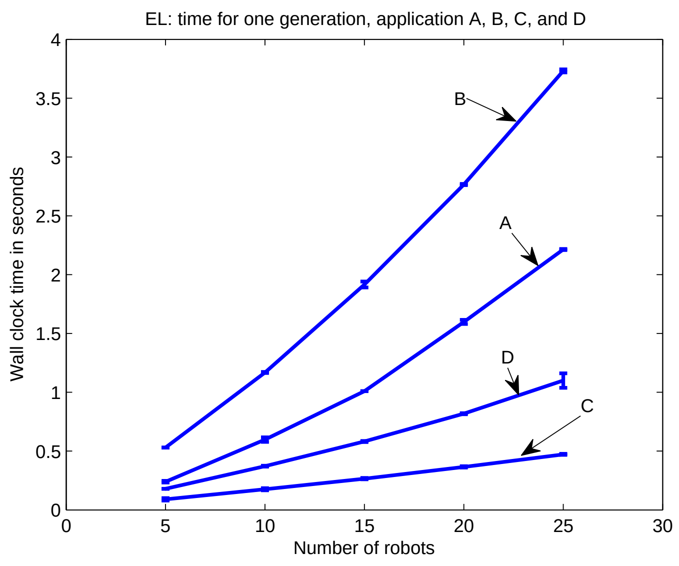
(a)

Figure 5.11: Comparison among ECA, CA, and RA of time required to generate first online solution for application D. RA does not appear on this graph because it is off the chart even for 5 robots. (Each data point in this figure is the average of five runs.)



(a)

Figure 5.12: Comparison among ECA, CA, and RA of time required to generate first online solution for application D, using a different scale than Figures 5.8, 5.9, and 5.10. CA cannot find solutions for teams of size ≥15; RA cannot find solutions for teams of size ≥20. (Each data point in this figure is the average of five runs.)



(a)

Figure 5.13: EL: Time to evolve one evolutionary generation for applications A, B, C, and D. (Each data point in this figure is the average of five runs.)

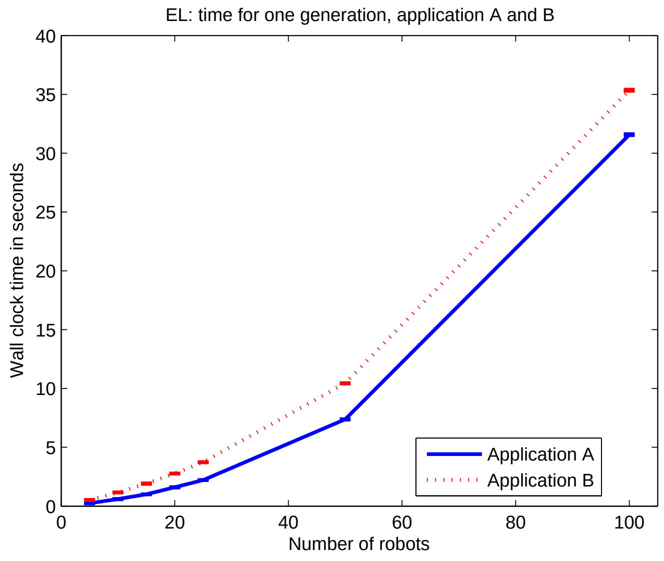
the elapsed time for evolving new generations, and time for extracting first-level chunks from the EL solution. Figure 5.13 shows the time required to evolve one generation for applications A, B, C, and D. The simulation results show that application A and B require more time to evolve one generation. In order to further explore the growth of the time for one generation, additional simulations are conducted for a team of 50 and a team of 100 robots for application A and application B. Figure 5.14 shows the result, which indicates a polynomial tendency.

All four applications require almost no time (<0.01 second) for the harvesting process for team of 5 to 25 robots, as shown in Figure 5.15.

5.3.3 Quality of solutions generated by chunking

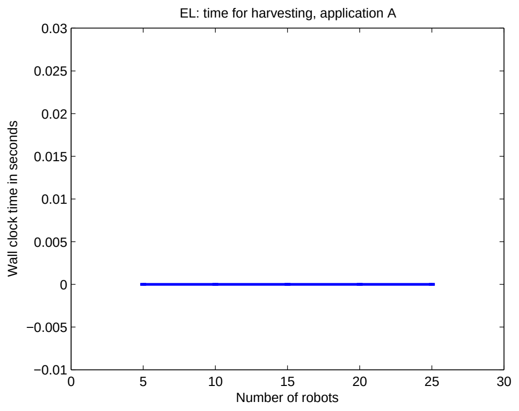
There are two solutions generated by chunking. The EL solution is generated off-line by the EL process. The EL solution is used to extract chunks, which are used in the ECA process to generate online team solutions. In this section, both solutions are examined.

Figure 5.16 shows example simulation results of EL for application A with 25 robots. In this example, the costs and complexity first increase, then decrease over time during the search, while the number of robots that can achieve a goal and the solution fitness increases. Although EL shows improvement of the solutions over time, various parameter settings may affect the end results of the EL process. It is up to the developer to find the best set of parameter settings for specific applications. While the fitness value and the number of successfully assigned robots increase over time, the complexity and the costs show irregular behavior.

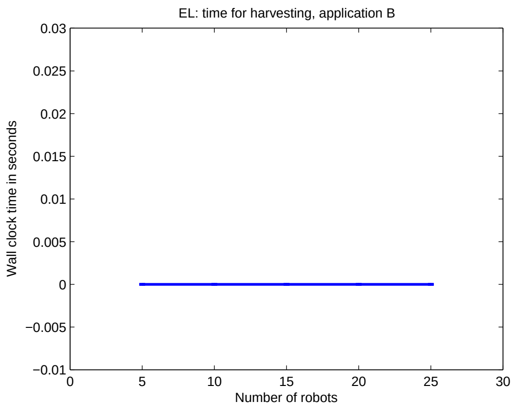


(a)

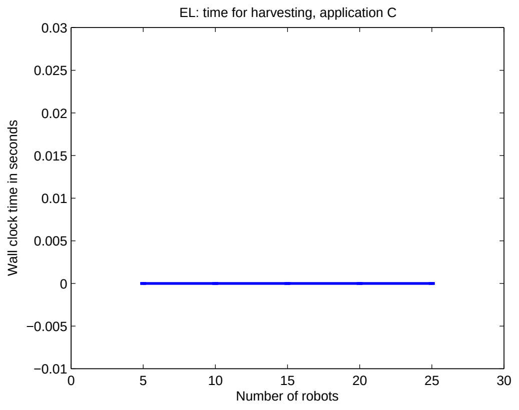
Figure 5.14: EL: Time for one evolutionary generation for application A and application B. (Each data point in this figure is the average of five runs.)



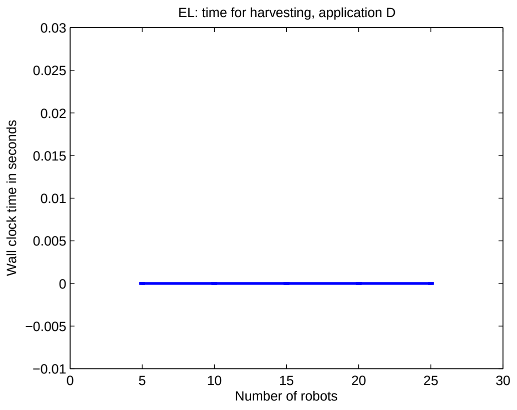
(a)



(b)

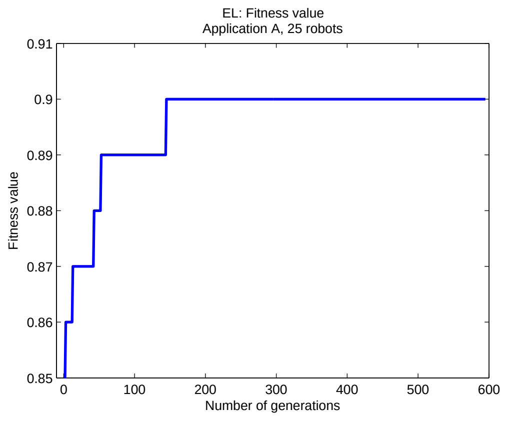


(c)

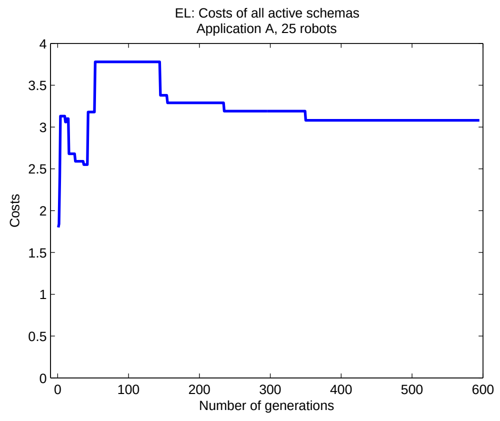


(d)

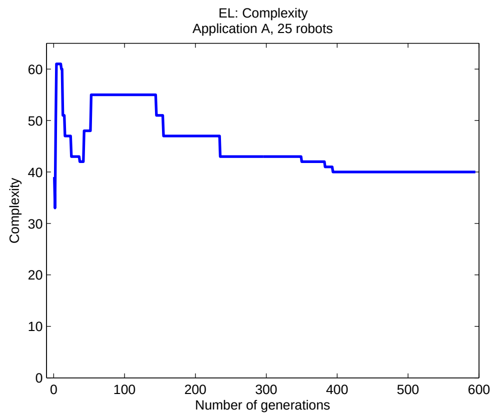
Figure 5.15: Time requirements of the chunking process, for harvesting first level chunks and to generate second-level chunks based on the harvested first-level chunks. It is measured within the resolution of 0.01 second. The values appear to be 0, because they are <0.01 second. (Each data point in these figures is the average of five runs.)



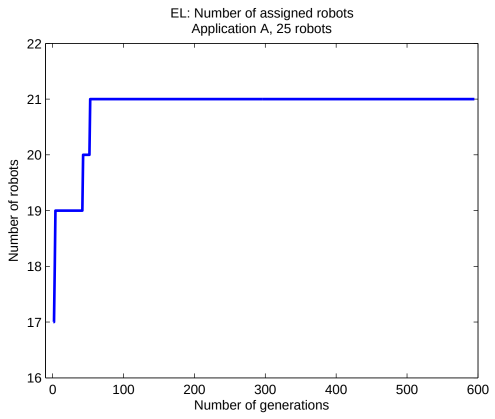
(a)



(b)



(c)



(d)

Figure 5.16: For a team of 25 robots and application A, these graphs show, during the EL process, the change over time of: (a) Fitness value; (b) Cost; (c) Complexity; (d) Number of robots that can accomplish the task. (Each data point in these figures is the result from one run that is representative of the typical behavior of this algorithm.)

Figure 5.17 shows another example simulation results of EL for application B with 25 robots. In this example, the costs and complexity oscillate for the first 100 generations during the search. The number of the robots that can achieve a goal also oscillates for the first 100 generations while the solution fitness increases.

The simulation results show that EL is able to generate solutions with increasing fitness value. The fitness value is a weighted combination of the costs of all active schemas, complexity, i.e. connectivity between the schemas, and goal achievement. In the original algorithm design, goal achievement covers two aspects: 1) the number of robots that are assigned successfully to perform the task, i.e. all the required information types for these robots can be fulfilled; and 2) the number of fulfilled information types that are required by the task definition. However, the number of fulfilled information types did not show any effect on improving the final results. One possible explanation is that the first aspect has much more significance than the second aspect. Hence, in the fitness calculation process, the default weight value is set to zero for the second aspect of goal achievement.

In order to calculate a fitness value in the range of 0 to 1, normalization is used. In order to normalize complexity and costs, the user needs to estimate the maximum number of connections between schemas in the graph, as well as the maximum costs of all active schemas. It is a balancing act to provide the maximum values that are not too high, lest the normalized value is too small to measure the difference between two solutions. If the provided maximum values are too low, the program will announce the current maximum values so that the user can update the parameter settings.

Although EL does not always converge to a complete solution, the chunks extracted from the EL solution, partial or complete, can be used to generate a complete team task solution in ECA, the online solution search process. Figure 5.18 shows the number of robots that can be assigned in the off-line EL solution and in the online ECA solution. The simulation results show that the default parameter settings do not ensure a complete solution for application C and application D. Figures 5.19 and 5.20 show the result of different parameter settings that deliver complete solutions. In Figure 5.19, both the inter-robot connection rate and the intra-robot connection rate are set to 0.95, instead of the default value of 0.8. In Figure 5.20, the intra-robot connection rate is set to 1.

5.3.4 Comparison of the solution quality generated by CA, RA, and ECA

CA, RA, and ECA of the assimilation process can deliver complete solutions for robot teams of 5 to 25 robots for applications A, B, and C. CA and ECA deliver the same solution, while RA normally delivers a solution with higher costs. Figures 5.21, 5.22, and 5.23 show the results for applications A, B, and C.

Application D poses a challenge for CA and RA. Because of the nature of the limited resource requirements, a greedy search can only find the solution for specific sequences of robots. Because the heuristics of CA are designed to search all small solutions first, and because the number of possible solutions is exponential in the number of robots, CA is not able to deliver a solution for a team of 15 robots after 2 days of continuous running time (on typical present-day Linux PC machines). The RA approach is able to find a solution after 20 minutes for a team of 15 robots for application D, but it is not able to deliver a solution in a timely manner for 20+ robots. The chunking approach can find solutions for all teams of 5 to 25 robots for application D, and can find solutions faster than CA and

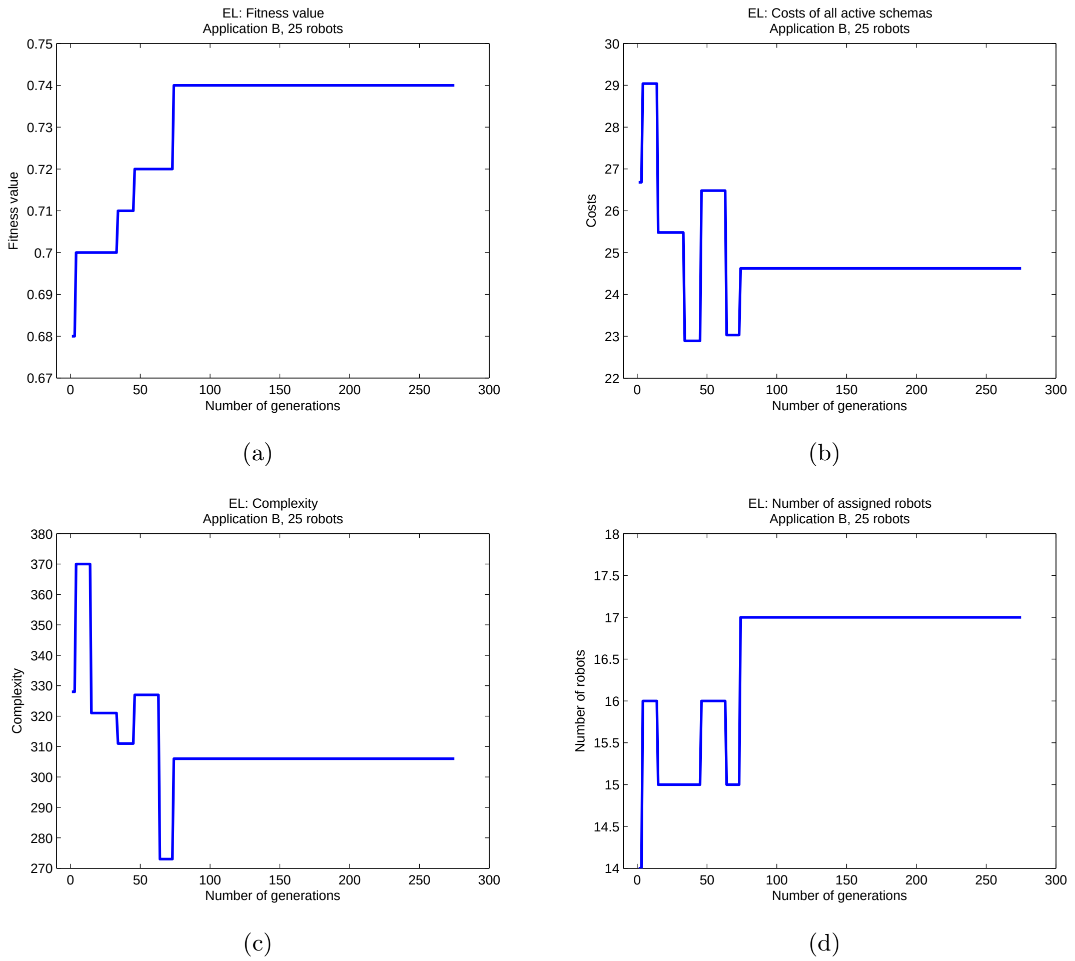
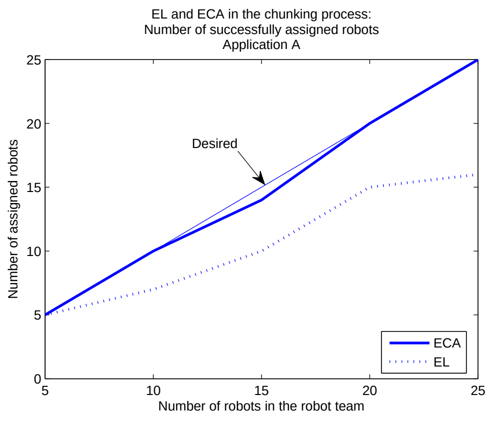
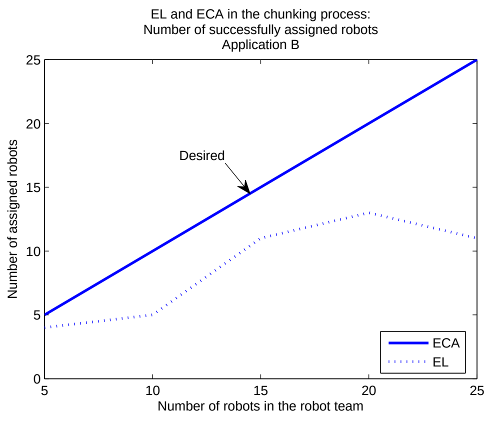


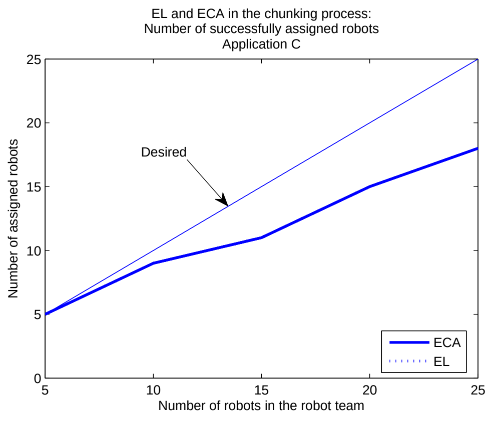
Figure 5.17: These graphs show, for a team of 25 robots and application B, during the EL process, the change over time of: (a) Fitness value; (b) Cost; (c) Complexity; (d) Number of robots that can accomplish the task. (Each data point in these figures is the result from one run that is representative of the typical behavior of this algorithm.)



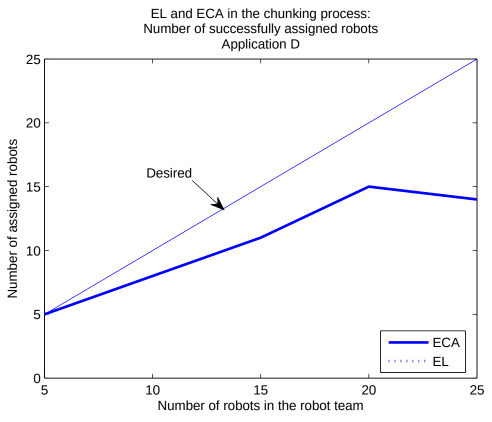
(a)



(b)

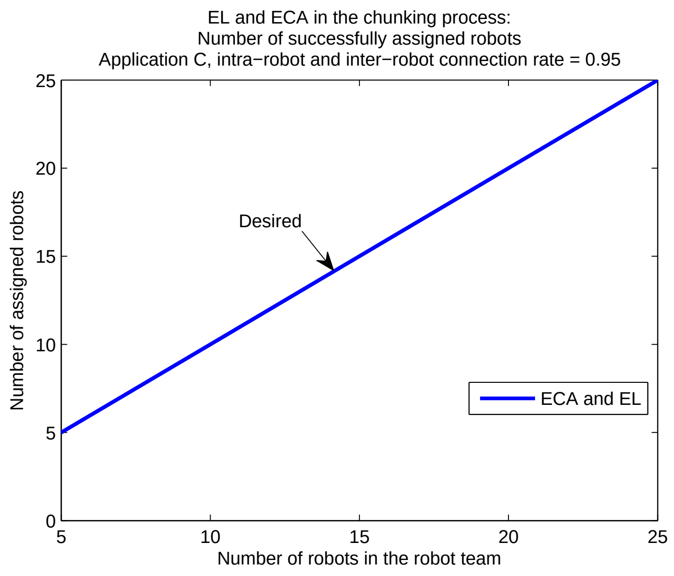


(c)



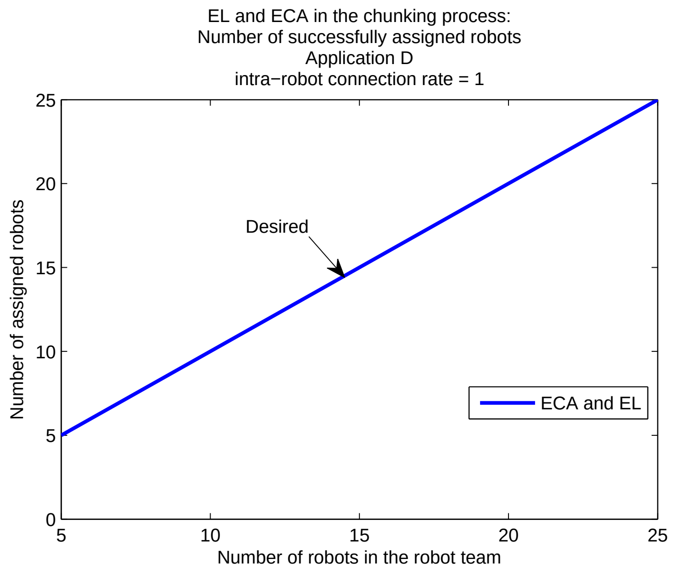
(d)

Figure 5.18: These graphs show the number of assigned robots in the off-line EL solution and in the online ECA solution for: (a) Application A; (b) Application B; (c) Application C; and (d) Application D. The inter-robot connection rate and the intra-robot connection rate are set to the default value of 0.8. For application C and application D, ECA and EL have the same value. (Each data point in these figures is the result from one run that is representative of the typical behavior of this algorithm.)



(a)

Figure 5.19: The number of assigned robots in the off-line EL solution and in the online ECA solution for application C. The inter-robot connection rate and intra-robot connection rate are set to 0.95. ECA and EL have the same value. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)



(a)

Figure 5.20: The number of assigned robots in the off-line EL solution and in the online ECA solution for application D. The inter-robot connection rate is set to the default value of 0.8. The intra-robot connection rate is set to 1. ECA and EL have the same value. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)

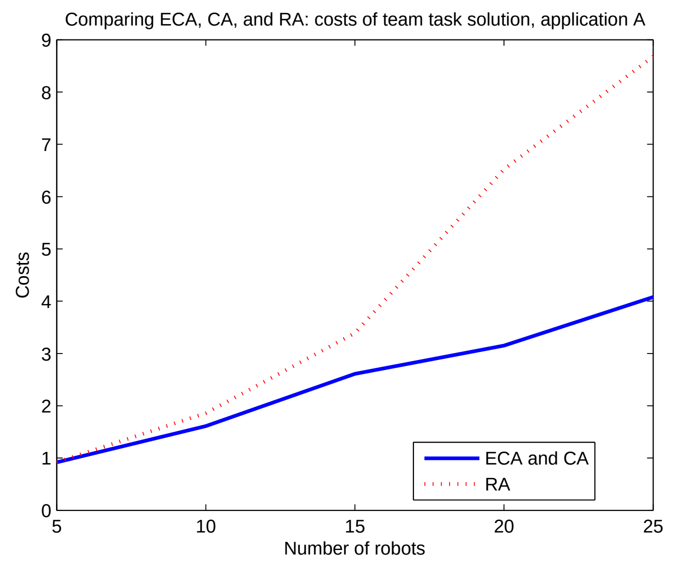


Figure 5.21: ECA, CA, and RA: Solution costs for application A. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)

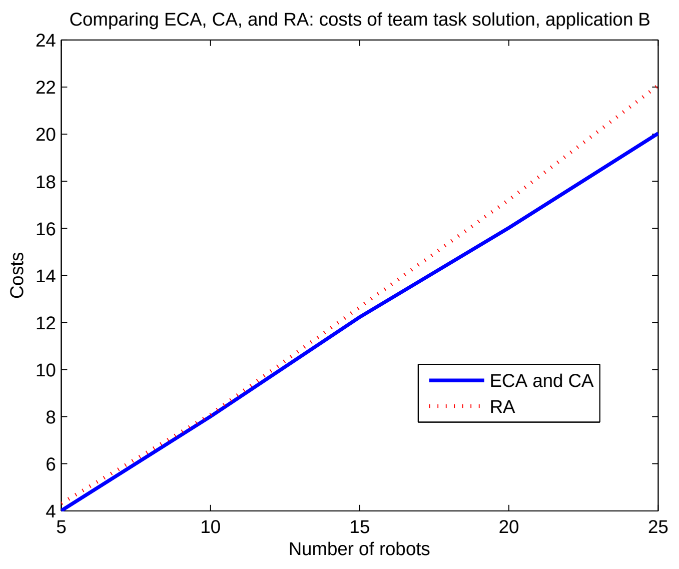


Figure 5.22: ECA, CA, and RA: Solution costs for application B. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)

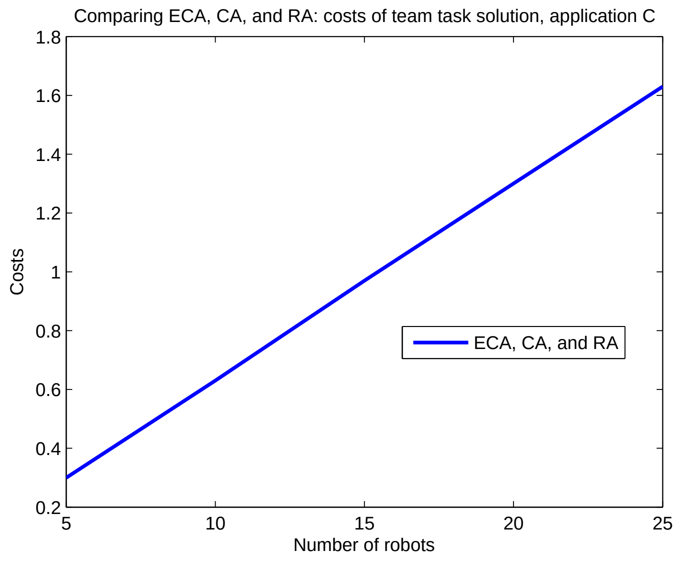


Figure 5.23: ECA, CA, and RA: Solution costs for application C. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)

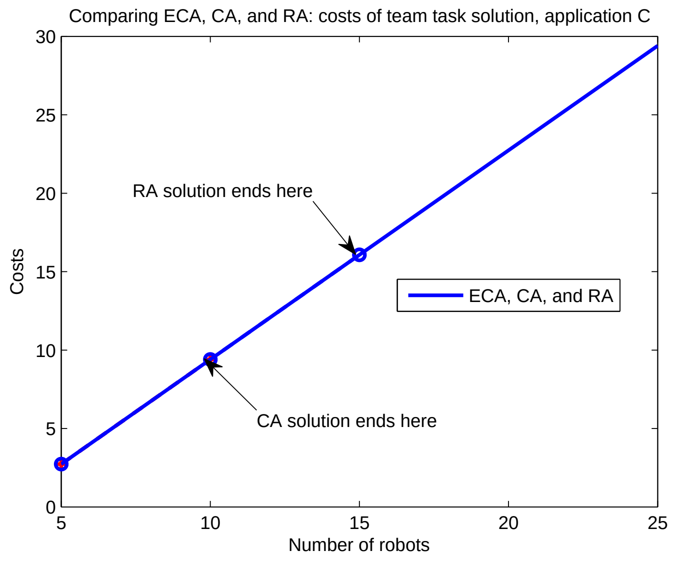


Figure 5.24: ECA, CA, and RA: Solution costs for application D. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)

RA. In the cases when CA and RA can produce solutions, they deliver solutions with equal quality to ECA. Figure 5.24 shows the results for applications D.

5.3.5 Comparison between chunking and ASyMTRe

ASyMTRe and chunking are different in the following ways:

- Chunking generates second-level chunks for each robot type, while ASyMTRe generates potential solutions for each individual robot;
- Chunking harvests and reuses first-level chunks, while ASyMTRe generates new potential solutions from scratch;
- Chunking uses EL to evolve highly fit chunks, while ASyMTRe either uses greedy search (CA) or randomized search (RA);
- Chunking provides more concrete team solutions including robot information and information flow information;
- Chunking can find solutions in cases when ASyMTRe (CA) cannot. However, often CA can generate a good solution fast.

ASyMTRe generates potential solutions for each robot team. The chunking process generates second-level chunks for each robot type in the robot team. The simulation results show that ASyMTRe generates many more potential solutions than the second-level chunks generated by chunking. The reason is that many potential solutions are not part of the solutions that EL generates off-line. In this case, the EL process works as an evaluation process to prune out the less relevant chunks. Chunks are less relevant if they are not often used in an actual solution. With a well-defined fitness function and the right parameter settings, EL can generate solutions and partial solutions with meaningfully high fitness values. From these solutions and partial solutions, more relevant chunks can be extracted.

5.3.6 Sensitivity tests of EL

EL generates different solutions using different parameter settings. Figures 5.25 to 5.28 show the simulation results of EL for different connection rate settings. While the inter-robot connection rate and/or the intra-robot connection rate range from 0.1 to 1, all the other parameters are set to default values. These results suggest that the best values for both the inter-robot connection rate and the intra-robot connection rate are 0.9 for an arbitrary, previously unknown application, in order to achieve the highest number of assigned robots.

Figure 5.29 shows the simulation results of EL for different weight settings for fitness calculation. While the weight for complexity increases from 0 to 0.8 and the weight for the number of assigned robots decreases from 0.8 to 0, all the other parameters are set to default values. These results show that EL is sensitive to different parameter settings. These results suggest that for an arbitrary application, when the weight of costs is 0.2, the best value for the weight of complexity is 0.1, and the best value for the weight of the number of assigned robots is 0.7, in order to achieve the highest number of assigned robots.

Figures 5.30 to 5.32 show results of sensitivity tests of EL for a changing mutation rate. While the mutation rate ranges from 0 to 0.1, all the other parameters are set to default values. The simulation results show that there is a difference between mutation rate equal to 0 and mutation rate greater than 0. However, this difference is not always preferable. For example, Figure 5.32 shows that the number of assigned robots decreases with a mutation rate greater than 0 for application A for a team of 10 robots, for application B for a team of 5 to 20 robots, and for application D for a team of 15 to 25 robots.

One potential problem for EL is over-learning. The default setting for the maximum number of generations (MAX_GENERATION) is 100, while the default setting for the maximum number of generations without improvement (MAX_NO_IMPROVEMENT) is 20. To test the sensitivity of chunking toward the number of generations, these parameter values are set to 1000 and 200, respectively. The EL process will stop if the number of generations exceeds 1000, or the number of generations without improvement exceeds 200, whichever comes first. Figure 5.33 shows the simulation results for the number of generations with the new parameter setting. Figure 5.34 shows the maximum number of generations without improvement, before a new, improved generation had evolved. The number of generations without improvement exceeds the default setting 20 in most of the cases for application A and application B. However, does an increased number of generations improve the end result? Figure 5.35 compares the results with default parameter settings and an increased number of generations. Based on the results, improvement can be observed for Application A, but not for Application B. For both application A and B, the difference between the two sets of ECA results is very small, while the difference between the two sets of EL results is larger. This result confirms that chunking is able to extract useful chunks from EL solutions for online solution search.

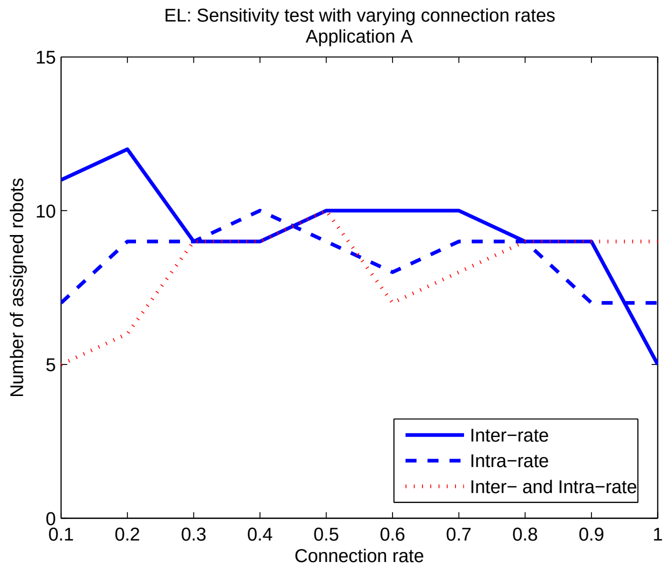


Figure 5.25: This graph shows the simulation results of the number of assigned robots in the solutions generated by EL for application A, for a team of 15 robots. The solid blue line shows the simulation results for the inter-robot connection rate, which ranges from 0.1 to 1, while the intra-robot connection rate has the constant default value of 0.8. The dashed blue line shows the simulation results for the intra-robot connection rate, which ranges from 0.1 to 1, while the inter-robot connection rate has the constant default value of 0.8. The dotted red line shows the simulation results for the inter-robot connection rate and the intra-robot connection rate, both ranging from 0.1 to 1. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)

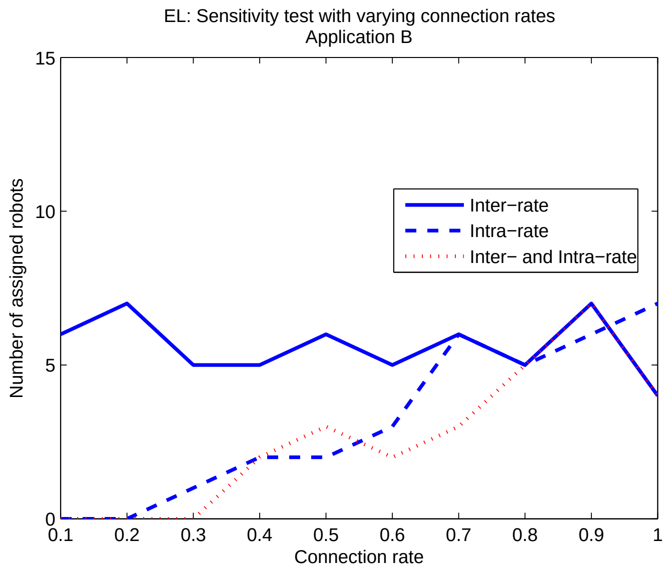


Figure 5.26: This graph shows the simulation results of the number of assigned robots in the solutions generated by EL for application B, for a team of 15 robots. The solid blue line shows the simulation results for the inter-robot connection rate, which ranges from 0.1 to 1, while the intra-robot connection rate has the constant default value of 0.8. The dashed blue line shows the simulation results for the intra-robot connection rate, which ranges from 0.1 to 1, while the inter-robot connection rate has the constant default value of 0.8. The dotted red line shows the simulation results for the inter-robot connection rate and the intra-robot connection rate, both ranging from 0.1 to 1. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)

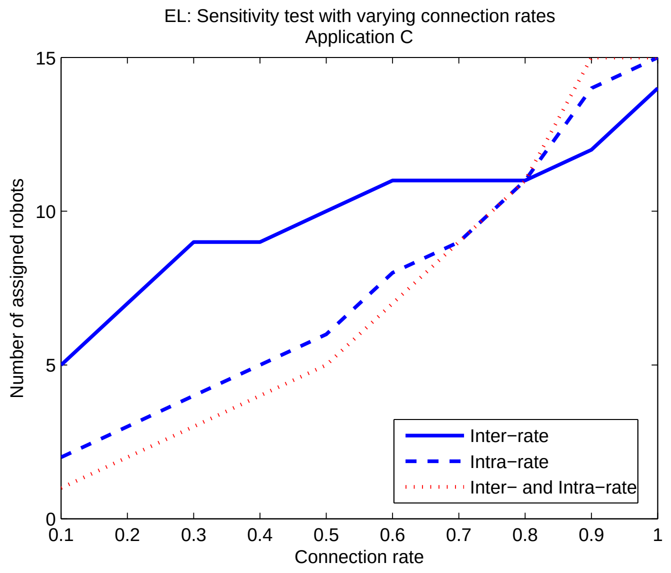


Figure 5.27: This graph shows the simulation results of the number of assigned robots in the solutions generated by EL for application C, for a team of 15 robots. The solid blue line shows the simulation results for the inter-robot connection rate, which ranges from 0.1 to 1, while the intra-robot connection rate has the constant default value of 0.8. The dashed blue line shows the simulation results for the intra-robot connection rate, which ranges from 0.1 to 1, while the inter-robot connection rate has the constant default value of 0.8. The dotted red line shows the simulation results for the inter-robot connection rate and the intra-robot connection rate, both ranging from 0.1 to 1. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)

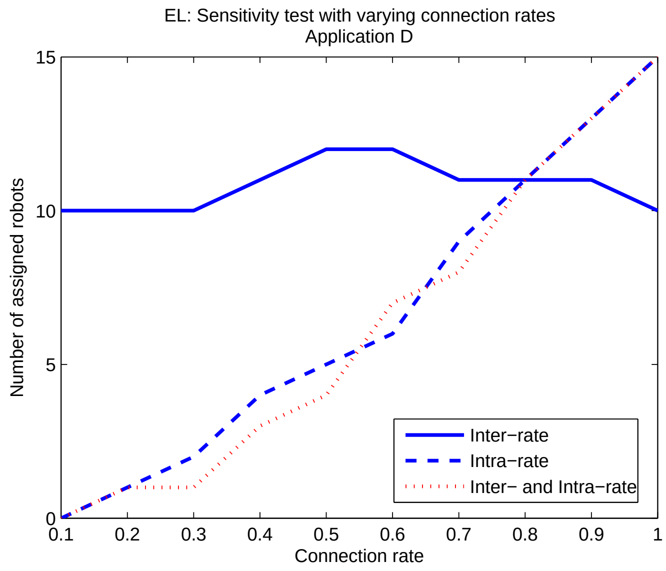
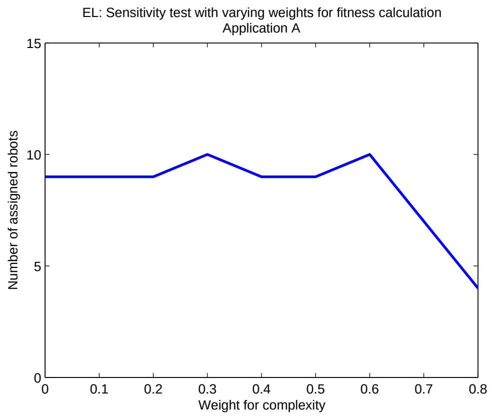
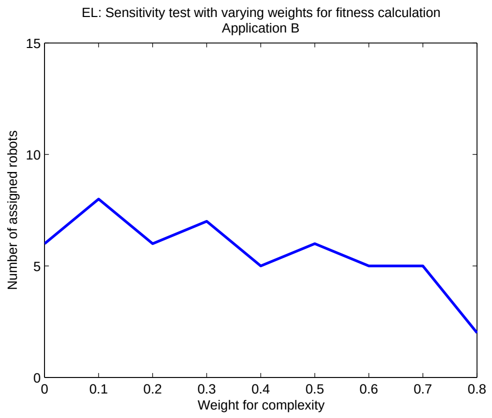


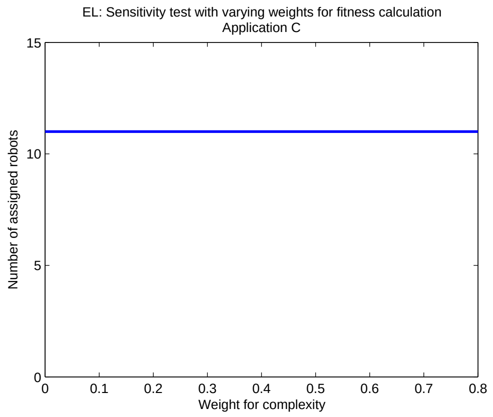
Figure 5.28: This graph shows the simulation results of the number of assigned robots in the solutions generated by EL for application D, for a team of 15 robots. The solid blue line shows the simulation results for the inter-robot connection rate, which ranges from 0.1 to 1, while the intra-robot connection rate has the constant default value of 0.8. The dashed blue line shows the simulation results for the intra-robot connection rate, which ranges from 0.1 to 1, while the inter-robot connection rate has the constant default value of 0.8. The dotted red line shows the simulation results for the inter-robot connection rate and the intra-robot connection rate, both ranging from 0.1 to 1. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)



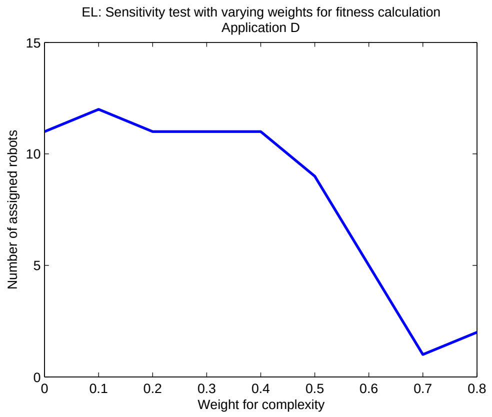
(a)



(b)

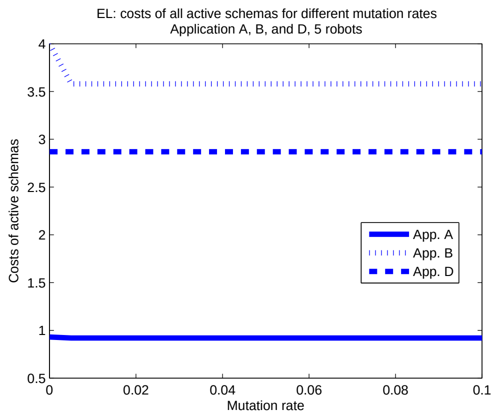


(c)

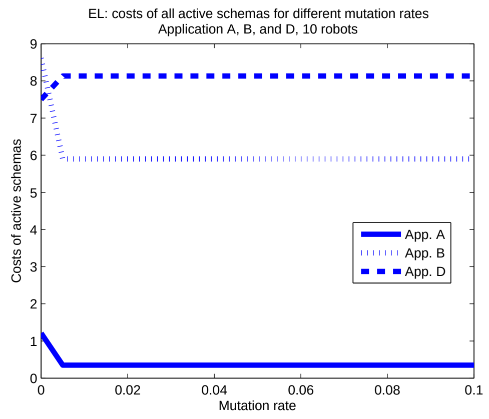


(d)

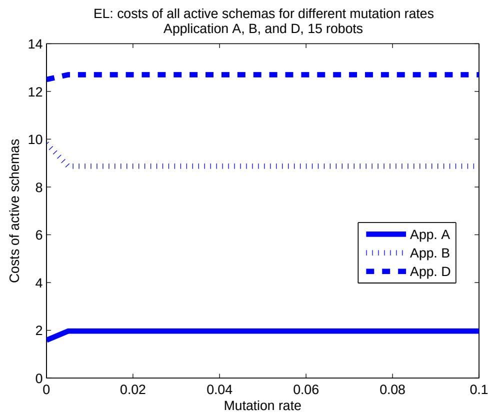
Figure 5.29: These graphs show the simulation results of the number of assigned robots in the solutions generated by EL for a team of 15 robots, for applications A, B, C, and D: (a) Application A; (b) Application B; (c) Application C; (d) Application D. Recall that the fitness function is: $F = w_c \cdot (c/c_{max}) + w_x \cdot x + w_q \cdot (q/q_{max}) + w_u \cdot (u/n)$.
In these simulations, the weight for costs(w_c) has the constant default value of 0.2, and the weight for the number of fulfilled information types(w_q) has the constant default value of 0. The weight for complexity(w_x) varies from 0 to 0.8, and is shown in the graph. The weight for the number of assigned robots(w_u) also varies from 0 to 0.8. The value of w_u is calculated using $w_u = 1 - 0.2(w_c) - w_x$, so that all the weights sum up to 1. The graphs show the results of increasing w_x and decreasing w_u. (Each data point in these figures is the result from one run that is representative of the typical behavior of this algorithm.)



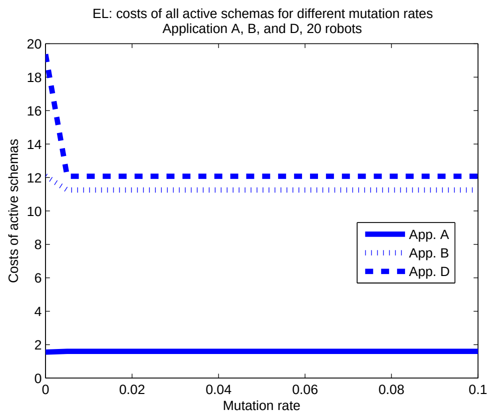
(a)



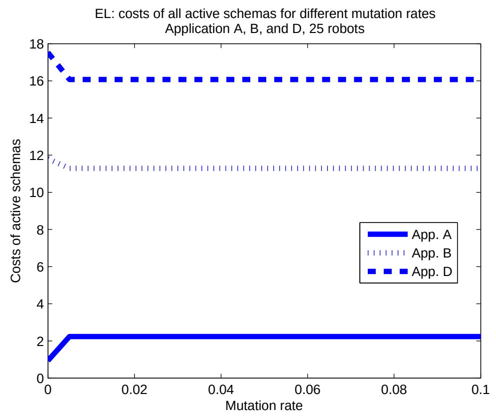
(b)



(c)

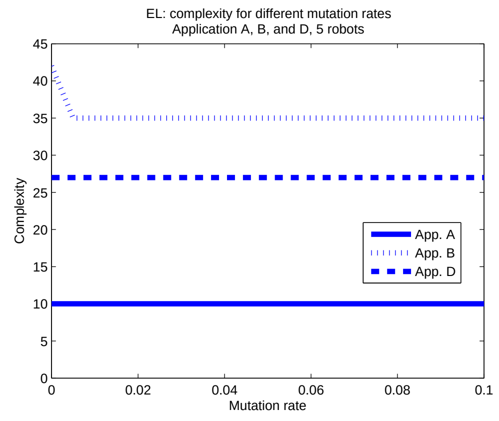


(d)

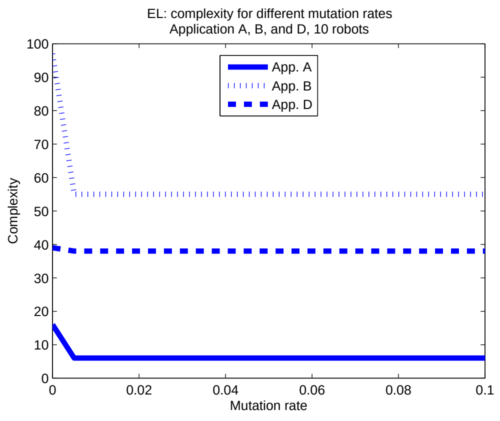


(e)

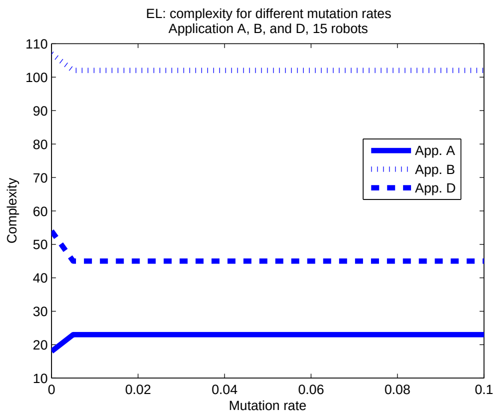
Figure 5.30: Costs for all active schemas in the EL solutions for different mutation rates. (Each data point in these figures is the result from one run that is representative of the typical behavior of this algorithm.)



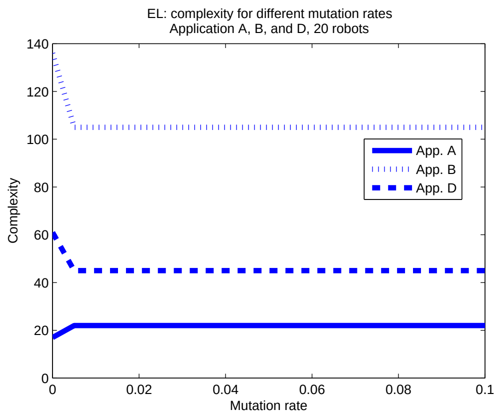
(a)



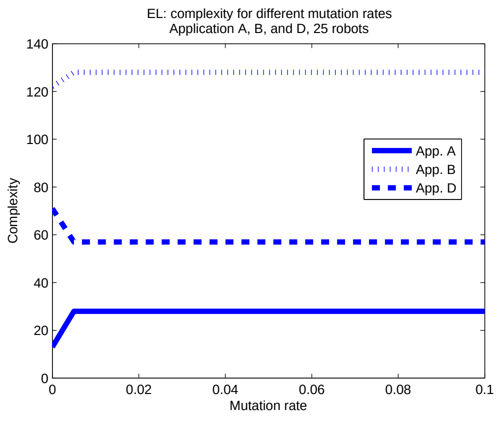
(b)



(c)

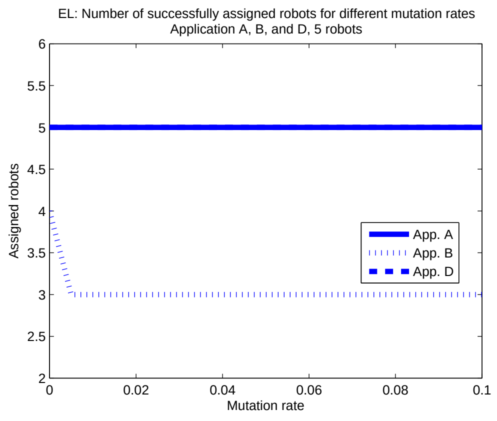


(d)

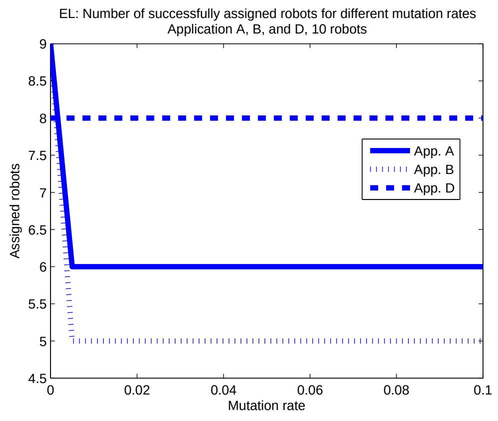


(e)

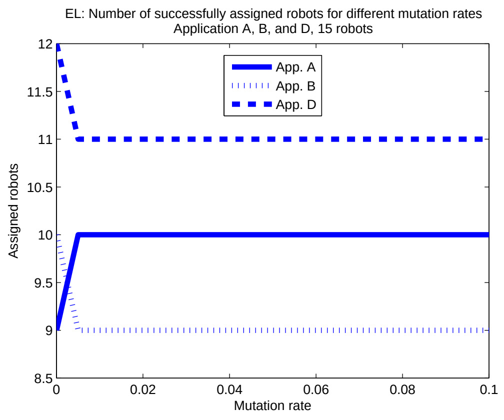
Figure 5.31: Complexity of the EL solutions for different mutation rates. (Each data point in these figures is the result from one run that is representative of the typical behavior of this algorithm.)



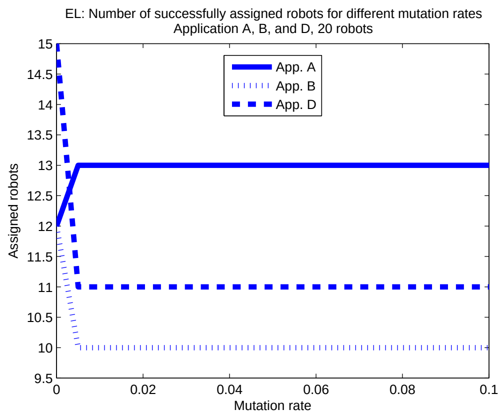
(a)



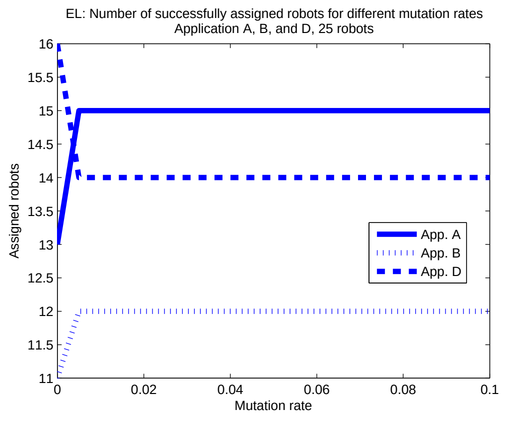
(b)



(c)



(d)



(e)

Figure 5.32: Number of assigned robot in the EL solutions for different mutation rates. (Each data point in these figures is the result from one run that is representative of the typical behavior of this algorithm.)

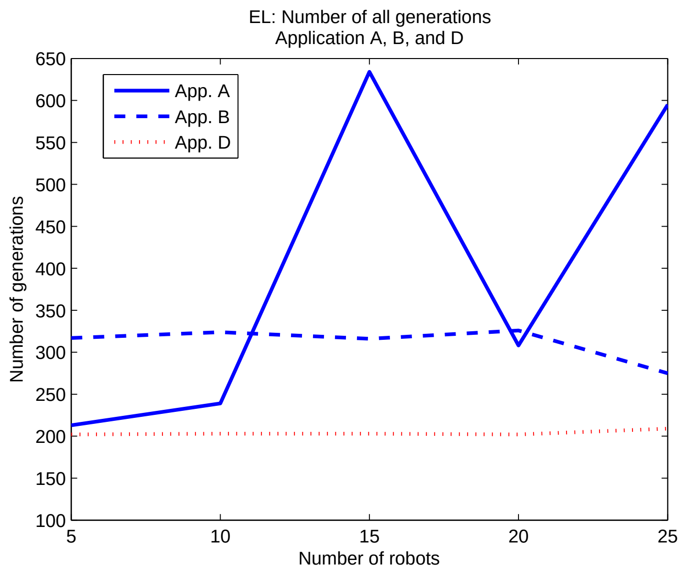


Figure 5.33: Number of generations in the EL process with new parameter setting: MAX_GENERATION = 1000 instead of 100, MAX_NO_IMPROVEMENT = 200 instead of 20. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)

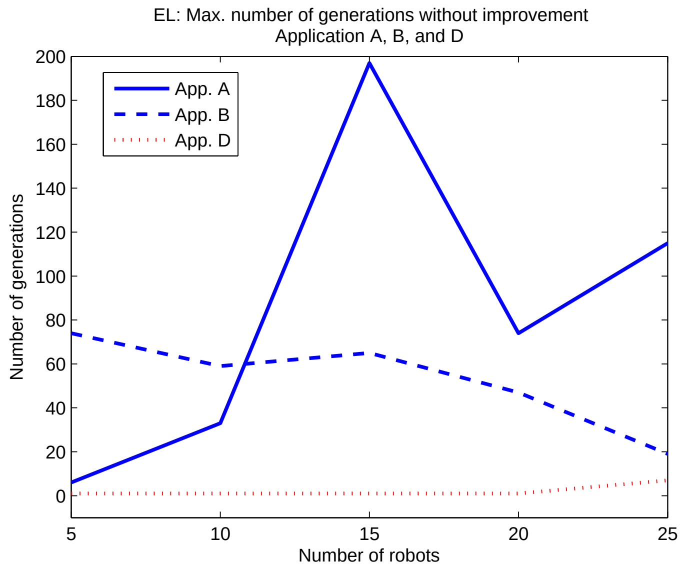
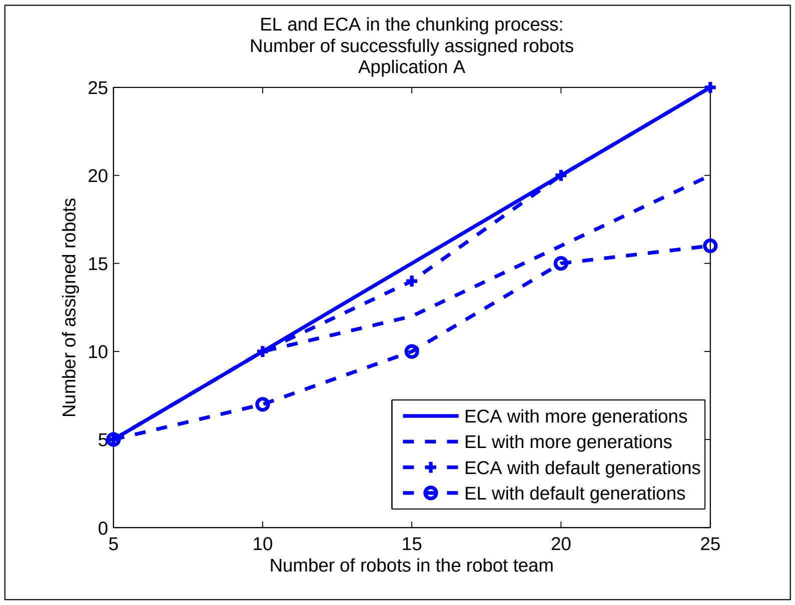
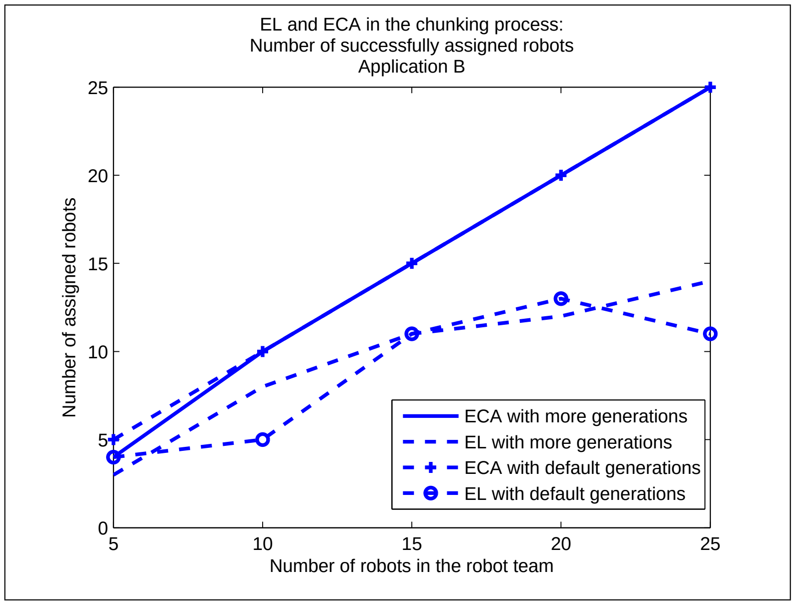


Figure 5.34: Number of generations with no improvement in the EL process with new parameter setting: MAX_GENERATION = 1000 instead of 100, MAX_NO_IMPROVEMENT = 200 instead of 20. The EL process will stop if the number of generations exceeds 1000, or the number of generations without improvement exceeds 200. (Each data point in this figure is the result from one run that is representative of the typical behavior of this algorithm.)



(a)



(b)

Figure 5.35: Number of assigned robots for different parameter settings of MAX_GENERATION (1000 and 100) and MAX_NO_IMPROVEMENT (200 and 20): (a) Application A, chunking with higher number of generations can assign more robots both in the EL and in the ECA process for a team of 10 robots, and for a team of 15 robots; (b) Application B, chunking with higher number of generations can assign more robots in EL in the most cases. However, chunking with higher number of generations can assign less robots in ECA for 5 robots. (Each data point in these figures is the result from one run that is representative of the typical behavior of this algorithm.)

5.4 Summary of Findings

In all the simulated applications, the online search performance of ECA is always faster than CA and RA. For the more complicated application D, ECA finds solutions in the same small amount of time, while the time requirement for CA and RA grows exponentially. All three approaches require some pre-processing. While CA and RA perform pre-processing online, chunking performs pre-processing off-line. Chunking needs more pre-processing time than CA and RA for applications A, B, and C. In relatively simple applications like applications A, B, and C, the pre-processing time is insignificant, but for a more complicated application like application D, the benefit of the chunking approach becomes apparent, because the pre-processing time of chunking is still less than a second, while it increases exponentially for CA and RA.

The time requirement for the off-line chunking process includes evolving new generations and harvesting chunks. The time requirement for new generations increases quadratically, while the time requirement for harvesting is insignificantly small.

The simulations also show that EL is able to increase the solution fitness over time. It is not always able to produce a complete solution off-line, but it is able to produce highly-fit chunks, which, when used in the ECA process, can sometimes produce a complete online solution. CA and ECA produce similar solutions for the simpler applications A, B, and C, while RA normally delivers a solution with higher costs. For application D, in the cases when CA and RA can produce solutions in a timely manner, they deliver solutions with equal quality to ECA.

The simulations show that ASyMTRe generates many more potential solutions than the number of second-level chunks generated by EL, because EL is using a filter to prune out the less relevant chunks. During the simulation, parameter adjustment is needed for application D, while the default parameter settings seem to work well for simpler applications. After adjusting the parameters, it is possible to find a complete team solution. The knowledge of proper parameter settings is a limitation of the chunking approach. It depends on the user's expertise to determine how the parameters should be set.

In summary, none of the approaches among CA, RA, and chunking always outperforms the other ones. The simulation results suggest that chunking may be preferable for more complex problems that involve more resource constraints, while CA and RA may be preferable for simpler applications that involve fewer resource constraints. The simulation results show that chunking is able to produce useful chunks that significantly increase the efficiency of the online search process versus the CA approach, i.e., using the chunks, ECA finds an initial team task solution faster than CA in all four applications. For simple tasks, the time to find an initial team task solution is comparable for chunking and RA, but chunking produces lower cost solutions. Hence for tasks where the chunks can be re-used, chunking will most likely always be preferable to CA and RA. However, for tasks where new chunks need to be generated, it may be best to run the CA and chunking algorithms concurrently, since it is difficult to determine in advance whether chunking's faster online search performance will outweigh its greater start up time in finding chunks. RA need not be included in a hybrid approach because it generates higher cost solutions than either CA or chunking, and while it has a considerable relative speed advantage over CA, its speed in absolute terms is not sufficient to warrant running all three approaches in parallel.

Overall, chunking holds considerable promise as an approach for assigning tasks to robots. First, its excellent online search performance suggests that it is producing high-quality chunks, and hence is a promising approach for continuous learning. Second, its excellent performance on the more complex task, coupled with the fact that robots will be asked to perform increasingly complex tasks in the future, suggest that ultimately, combining off-line chunking and online search may be a more viable approach than online search alone.

Chapter 6

Conclusions and Future Work

In this dissertation, I have laid out a Schema-Based Constructivist Robot Learning Architecture (SB-CoRLA) that is closely related to the ASyMTRe algorithm. I have identified different learning processes in SB-CoRLA for enabling learning through Assimilation and Accommodation, and implemented the Assimilation process. Specifically, I have developed three new algorithms:

1. An off-line algorithm (EL) that uses a genetic algorithm to develop highly-fit robot team solutions;

2. An off-line harvesting algorithm that extracts first-level chunks from the highly-fit off-line robot team solution; and

3. An online solution search algorithm (ECA) that combines first-level chunks to generate second-level chunks, as well as to assign the second-level chunks to a team of robots.

Accommodation is not implemented in this dissertation.

ASyMTRe automatically generates task solutions based on robot capabilities in the form of schemas. It generates solutions by connecting schemas via matching information types. However, it does not learn from past experience and restarts the search process from scratch every time a solution is needed. SB-CoRLA builds on the groundwork of schema-based robot system. It implements a genetic algorithm in off-line evolutionary learning (EL) to learn useful knowledge about the schema connections, called “chunks”, that are part of a highly-fit solution. In EL, each solution is an individual in a population and is represented as a graph. The vertices represents schemas, while the edges represent connections between schemas via matching information types. Throughout the EL process, new solutions are generated by recombining subgraphs from existing solutions via crossover, and by changing connections between the vertices via mutation. A fitness function is used to measure the fitness of the solution and is defined as weighted sum of the number of assigned robots, the complexity which is the number of connections between schemas, and the costs of all active schemas. In the harvesting process, first-level chunks are generated by backtracking each complete information flow that provides an information type that is required by the task definition. In the online solution search process (ECA), relevant first-level chunks are selected based on the robot team configuration and the task definition. These first-level chunks are combined into second-level chunks, which represent complete solutions for each robot type. ECA then assigns the second-level chunks to the robots to generate a robot team solution. ECA iterates through robot types in the robot team. CA iterates through the robot IDs in the robot team. ECA and CA both have exponential time complexity. CA generates the sequential ordering of the robots based on their IDs ($O(n!)$), while ECA generates the sequential ordering of the robots based on their types ($O(\frac{n!}{t_1!t_2!...t_i!...t_m!})$, where t_i is number of robots of robot type i in the robot team configuration).

To evaluate the efficiency of the algorithms, I implemented four simulated tasks and ran various simulations to test ECA with teams of 5 to 25 robots, and to test EL and harvesting with teams of 5 to 100 robots. The simulation results suggest that in practice EL performs quadratically, while harvesting essentially requires constant time, despite the fact that, theoretically, big-O performance of harvesting is linear in the number of edges of the graph. ECA requires insignificantly little time to generate a first online solution, but the big-O performance of ECA for all iterations of robot types is exponential. The simulation results also suggest that the pre-processing time for EL and harvesting are competitive with the pre-processing time for the CA and RA algorithm, and that for more complicated tasks, EL may significantly outperform CA and RA algorithm. Finally, ECA consistently finds an initial solution more quickly than CA and RA. The simulation results seem to confirm the viability of the implemented assimilation process in the framework of SB-CoRLA.

As such, this dissertation provides a foundation for continuous learning in a schema-based robot system. In order to realize its potential in the future, there are several promising research directions to be explored:

- Developing SCS repository with an indexing system: In this dissertation, the number of chunks is small enough that searching through all the chunks is quite efficient. As the size of the repository grows, it may no longer be feasible to completely search the repository to find the relevant chunks. In this case, it is essential to have an indexing system that will retrieve the most relevant chunks. Furthermore, this indexing system can also be used to prune the SCS repository and to clean out less relevant chunks.

- Gaining more insight for parameter settings: As the simulation results show, the parameter settings for EL can change the final outcome of the team solution. Extensive experimentation will probably be required to determine the correlation between certain parameter settings and application characteristics, in order to generate high quality solutions.

- Including human knowledge: The chunking process cannot deal with new robot types and new information types. Ordinarily, in order to integrate new robot types and new information types in the existing SCS repository, the EL process is needed. However, an experienced user might be able to find similar robot types and information types in the existing SCS repository and substitute the new with the old. It is also interesting to explore whether there are fully automated ways to perform this substitution process.

- Generating higher-level chunks: The current chunking process generates first-level chunks and second-level chunks. In order to increase search efficiency and to perform more sophisticated learning, it is important to explore algorithms that could create higher-level chunks. However, the complexity of such an algorithm could increase considerably.

- Implementing accommodation: Constructivist learning emphasizes the importance of both assimilation learning and accommodation learning. This dissertation has implemented the assimilation aspect of SB-CoRLA. Accommodation differs from assimilation in that it is an online learning process, rather than an off-line learning process. While assimilation learns new combinations of existing skills, accommodation learns new skills through modification of old skills by interacting with the environment. Therefore, accommodation presents a different set of challenges. A promising approach could be goal-directed feedback-based learning.

Appendix

Appendix A

Chunking Implementation Details

In this section I list some important implementation details.These details are implemented to make the chunking process, especially the evolutionary learning process (EL) cleaner, more understandable, and more efficient.

A.1 The Graph

A graph structure, in adjacency list format, is used to represent an individual team solution. In EL, each generation consists of a number of individual team solutions, i.e. a number of graphs. Each graph has as many nodes as the number of available schemas in the robot team. Each graph node is indexed with a unique integer. A *Graph_node* structure is used to define a node. The Graph_node structure contains the following information:

- The ID of the robot that this graph node belongs to;
- The type of the robot that this node belongs to. That robot type is determined by the available sensors on a robot;
- The name of the schema that this node represents;
- The costs of the schema that this node represents;
- A pointer to the schema that this node represents; and
- A list of communication input and output information types, if the node is a communication schema.

A *Vertex* structure is used to represent the connections in the graph. Each graph has two lists of vertices, one for the actual connections among the nodes, the other for backward tracing of the actual connections to increase computational efficiency. Each vertex is indexed with a unique graph node index, and maintains a list of *ChunkNode* structures to record information about other graph nodes that are connected to this graph node. The *ChunkNode* structure contains the following information:

- The graph node index;
- The ID of the robot that this node belongs to;

Table A.1: The Comm structure that represents a CS

Variable	**Description**
string input	input information type
string output	output information type
double prob	success probability of the schema
double cost	costs of the schema

- The type of the robot that this node belongs to;
- A boolean variable indicating whether or not this robot helps another robot; and
- A pointer to the schema that this node represents.

In addition to the nodes representing available schemas, there is another kind of node, called a *goal* node. The goal node represents the team task. It takes the required information types as inputs and does not provide any output. Each robot has its own goal node. When all inputs are satisfied for a robot's goal node, this robot is considered successfully assigned.

A.2 The Schema

There are two different schema types in the implementation: a) the CS, and b) the rest (PS and MS). CS has a different data structure and needs to be referred to differently than PS and MS. Table A.1 shows the content of the *Comm* structure that is used for CS. Table A.2 shows the content of the *Schema* structure that is used for PS and MS.

In SB-CoRLA, there are two kinds of CS:

- The schema *cs* on a helper robot transmits information to a robot that needs help; and
- The schema *cs2* on a robot that needs help receives information transmitted by a helper robot.

Both *cs* and *cs2* have one input and one output.

In the original ASyMTRe implementation, PS and MS can have different sets of multiple input information types and multiple output information types. In SB-CoRLA, PS and MS have a unique set of inputs and output. In the original ASyMTRe, PS and MS can have multiple outputs. In SB-CoRLA, PS and MS can have only one output. Those two changes are necessary to keep the graph structure for EL clean. Without those changes, the forward and backward tracing within the graph would be much more complicated and not clearly manageable. For example, if one schema can have two different sets of input information types, the corresponding graph node needs to keep track of these two sets of information types. In order to check whether or not all the required information types of this schema are provided, two separate tests need to be made, one for each set. It makes more sense to maintain two different schemas and two different corresponding graph nodes

Table A.2: The Schema structure that represents PS and MS

Variable	**Description**
string name	schema name
int id	unique id for each schema, to differentiate schemas with same name
int sensornum	number of sensors that are associated with this schema
vector<string> sensor	name of the sensors
int inputnum	number of input information types that this schema requires
vector<string> input	input information types
vector<string> output	output information type
double prob	success probability of the schema
double cost	costs of the schema

in this case, instead of one schema with two sets of information types. Similar consideration about multiple output information types leads to the design decision about having PS and MS with a single output information type.

A.3 The Procedures in EL

EL consists of initialization, selection, crossover, mutation, evaluation, and pruning procedures. One design decision is to record the most fit individual from each generation based on the fitness of a valid team solution, instead of the fitness of the individual "as is". Therefore the pruning procedure is created to generate valid team task solutions, either complete or partial, based on the population evolved throughout the EL process.

Both the evaluation and the pruning procedures identify incomplete information flows in a solution. An information flow consists of connections between schemas. An information flow is incomplete when not all required input information types are satisfied for all involved schemas. The evaluation process calculates the fitness value of a team solution without deleting incomplete information flows. Pruning deletes incomplete information flows and information flows that do not lead to a goal node. The tricky part is to backtrack the information flow, in order to determine whether an information flow is complete for providing an information type. While the evaluation procedure simply checks the validity of an information flow, the pruning procedure deletes connections between nodes when the information flow is not complete, even though the node connection is valid. For example, if $node_1 \rightarrow node_2 \rightarrow node_3 \rightarrow goal$ is a complete chain of nodes connected with each other to provide a certain information type required by the task, and $node_2 \rightarrow node_3 \rightarrow goal$ exists in the solution, the evaluation procedure calculates the fitness value for the team solution "as is", while the pruning procedure deletes the connection between $node_3$ and $goal$.

For each generation, two sets of populations are created[1]: a) a population before applying the pruning procedure, the un-pruned population, and b) a population after applying the pruning procedure, the pruned population. The pruned population is used to find the most fit team task solution in this generation. The un-pruned population is used for the selection, crossover, and mutation procedures to create a new generation. It is not enough to only use the pruned population because, although some information flow is incomplete, it could still be a part of a highly-fit, complete information flow chain. It is therefore necessary to use the un-pruned population in the EL process.

Since information type can be provided in different ways, there are sometimes redundant information flows in the final solution. A method in EL named *delete_alternate_ways* eliminates the redundancy randomly. It is not possible to perform greedy elimination, i.e., to eliminate the information flow with higher costs, because a node can be accountable to multiple information flows. During testing, I have found that sometimes it is better to turn off this method, since it can lead to no solution being found.

[1]A design decision is to perform pruning for every generation. However, the simulation results show that the fitness value of the best solution for each generation does not fluctuate over several generations. It is thus worth considering to reduce the frequency of pruning, e.g. to perform pruning once every 10 generations, in order to reduce the computational costs.

The user can use command line tags to vary many parameter settings for the EL process. Table A.3 shows the tag name, the parameter name and description, as well as their default settings, which are determined through testing.

In the following I explain the different EL parameters in detail:

- -n, NUM_R: The number of robots in a robot team configuration;
- -s, SUM_SCHEMA: The number of all available schemas in the robot team configuration; also the number of all nodes in the graph, including the goal node;
- -f, NEED_SUM_SCHEMA: In order to run the program for the EL process, the user needs to know the value of the parameter SUM_SCHEMA. One way to calculate it is to run the same program with this tag set to 1;
- -k, CONNECT_RATE: Determines how likely two schemas from the same robot with matching information types will be connected;
- -o, CS_CONNECT_RATE: Determines how likely two CSs from different robots with matching information types will be connected;
- -g, MAX_GENERATION: The EL process stops after evolving this number of generations;
- -j, MAX_NO_IMPROVEMENT: The EL process stops after the population has evolved this number of generations and the fitness value of the best solution in the population has not improved;
- -a, MSS: Defines how many robots can work together in a team;
- -b, MTH: Each robot can help a limited number of robots. This parameter defines the maximum number of robots that one robot can help;
- -p, POPULATION: The number of individuals, i.e., graphs representing task team solutions, in each population;
- -v, FP_NOT_TM: For the selection process and the crossover process, the user can choose between fitness proportionate selection and tournament selection. The same selection method will then be used for both selection and crossover. Previous experience has shown that tournament selection is appropriate when the user wants more diversity in the population;
- -w, TM_SELECTION_RATE: If the tournament selection is chosen, the user can use this tag to determine the probability of choosing the more fit solution out of two;
- -r, CROSSOVER_RATE: Determines how likely a crossover will take place;
- -m, MUTATION_RATE: Determines how likely a mutation will take place;

The following are weight factors for calculating the fitness value of a team task solution:

- -c, WEIGHT_COST: The weight for the costs of all active schemas;

Table A.3: EL command line tags, parameter names, parameter descriptions, and default settings

Tag	**Var. Name**	**Description**	**Default**
n	int NUM_R	number of robots	5.0
s	int SUM_SCHEMA	number of schemas	500.0
f	int NEED_SUM_SCHEMA	=1 if needed to calculate SUM_SCHEMA	0.0
k	double CONNECT_RATE	probability for intra-robot connection	0.8
o	double CS_CONNECT_RATE	probability for inter-robot connection	0.8
g	int MAX_GENERATION	max. number of generations	100.0
j	int MAX_NO_IMPROVEMENT	max. number of generations without improvement	20.0
a	int MSS	max. sub-team size	3.0
b	int MTH	max. number of robots that one robot can help	3.0
p	int POPULATION	number of individuals in each generation	500.0
v	int FP_NOT_TM	choose between fitness proportionate (=1) and tournament selection (=0)	1.0
w	double TM_SELECTION_RATE	probability of choosing the more fit solution in tournament selection	0.8
r	double CROSSOVER_RATE	probability of crossover	0.6
m	double MUTATION_RATE	probability of mutation	0.005
c	double WEIGHT_COST	for fitness calculation	0.2
x	double WEIGHT_COMPLEXITY	for fitness calculation	0.4
q	double WEIGHT_UTILITY	for fitness calculation	0.0
u	double WEIGHT_UTILITY2	for fitness calculation	0.4
d	double MAX_COST	max. cost for team solution	30.0
y	int MAX_COMPLEXITY	max. complexity for team solution	200.0
t	int MAX_UTILITY	max. utility for team solution	50.0
e	double MAX_SENSOR_COST	max. cost for a sensor	30.0

- -x, WEIGHT_COMPLEXITY: The weight for the number of all active connections between schemas;
- -q, WEIGHT_UTILITY: Recall that a task is defined as a set of required information types. This is the weight for the number of fulfilled required information types, i.e., complete information flows that lead to a required information type. The simulations have shown that this factor has less effect on the result than the others;
- -u, WEIGHT_UTILITY2: The weight for the number of robots that can provide all the required information types.

The following are variables that define the maximum values used to normalize different factors for calculating the fitness value of a team task solution:

- -d, MAX_COST: The maximum cost of all active schemas;
- -y, MAX_COMPLEXITY: Maximum number of active connections between schemas;
- -t, MAX_UTILITY: The maximum number of fulfilled required information types;
- -e, MAX_SENSOR_COST: The maximum cost for a single sensor. The maximum number of robots that can provide all the required information types is the same number as NUM_R, i.e., the number of robots in the robot team.

A.4 Preview of ECA v.2

The current ECA can only handle chunks. For a robot team composition and a task definition, ECA finds relevant first-level chunks from the SCS repository based on the robot types and the information types, and combine them into second-level chunks. In order for the chunking process to be able to use both basic schemas and chunks in the online solution search process, ECA v.2 is proposed for future work. This section lays out the basic steps of ECA v.2:

1. Parsing: Find relevant chunks and schemas based on robot team configuration and task definition;
2. Generating potential solutions: Create a list of potential solutions using both chunks and schemas while keeping a record of whether a potential solution contains chunks. Each potential solution is a graph in the form of adjacency list, which is very similar to the format of second-level chunks. One possible implementation is to first connect all valid connections in the initialization process of EL, then to extract all first-level chunks from the graph after pruning, and finally to combine the first-level chunks into second-level chunks. It might or might not be beneficial to keep track of robot types in this process. Furthermore, CA uses one list of potential solutions for all robots in the team; it is better to keep one list of applicable potential solutions for each individual robot based on their capabilities, i.e. available schemas;
3. Online solution search: Sort the potential solutions in ascending order of costs and perform a greedy search for all permutations of robot IDs. It is worth exploring

whether potential solutions containing chunks should be preferred, regardless of their costs.

Bibliography

Bibliography

[Arbib, 2003] Arbib, M. A. (2003). Schema theory. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pages 993–998. MIT Press, Cambridge, MA, USA.

[Arkin, 1987] Arkin, R. C. (1987). Motor schema based navigation for a mobile robot: an approach to programming by behavior. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 264–271.

[Arkin, 1998] Arkin, R. C. (1998). *Behavior-based robotics*. MIT Press.

[Barto and Mahadevan, 2003] Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379.

[Benjamin et al., 2004] Benjamin, D. P., Lyons, D., and Lonsdale, D. (2004). Adapt: A cognitive architecture for robotics. In *Proceedings 2004 International Conference on Cognitive Modeling*, Pittsburgh, PA.

[Bowling et al., 2004] Bowling, M., Browning, B., and Veloso, M. (2004). Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04)*.

[Brooks and Mataric, 1993] Brooks, R. A. and Mataric, M. J. (1993). *Robot Learning*, chapter 8. Kluwer Academic Publishers, Boston, Dordrecht, London.

[Bruner, 1990] Bruner, J. (1990). *Acts of Meaning*. Harvard University Press.

[Cambron and Peters, 2000] Cambron, M. and Peters, R. A. (2000). Sensory motor control for grasping in a humanoid robot. In *Proceedings 2000 IEEE International Conference on Systems, Man and Cybernetics*, Nashville, TN.

[Chaput, 2004] Chaput, H. H. (2004). *The constructivist learning architecture: A model of cognitive development for robust autonomous robots*. PhD thesis, The Department of Computer Sciences, The University of Texas at Austin.

[Chaput et al., 2003] Chaput, H. H., Kuipers, B., and Miikkulainen, R. (2003). Constructivist learning: a neural implementation of the schema mechanism. In *Proceedings of WSOM '03: Workshop on Self-Organizing Maps*, Kitakyushu, Japan.

[Cherubini et al., 2007] Cherubini, A., Giannone, F., Iocchi, L., and Palamara, P. (2007). An extended policy gradient algorithm for robot task learning. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4121–4126.

[Coelho and Grupen, 2000] Coelho, J. and Grupen, R. (2000). Learning in non-stationary conditions: a control theoretic approach. In *Proceedings 17th International Conf. on Machine Learning*, pages 151–158. Morgan Kaufmann, San Francisco, CA.

[Coradeschi and Saffiotti, 2003] Coradeschi, S. and Saffiotti, A. (2003). An introduction to the anchoring problem. *Robotics and Autonomous Systems. Special issue on perceptual anchoring*, 43(2-3):85–96.

[Deiterding and Henrich, 2007] Deiterding, J. and Henrich, D. (2007). Automatic adaptation of sensor-based robots. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1828–1833.

[Dogar et al., 2007] Dogar, M., Cakmak, M., Ugur, E., and Sahin, E. (2007). From primitive behaviors to goal-directed behavior using affordances. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 729–734.

[Donald, 1995] Donald, B. R. (1995). On information invariants in robotics. *Artificial Intelligence*, 72(1-2):217–304.

[Donald et al., 1994] Donald, B. R., Jennings, J., and Rus, D. (1994). Analyzing teams of cooperating mobile robots. Technical Report TR94-1429, Dept. of Computer Science, Dartmouth College.

[Donald et al., 1997] Donald, B. R., Jennings, J., and Rus, D. (1997). Information invariants for distributed manipulation. *The International Journal of Robotics Research*, 16(5):673–702.

[Doumont, 2002] Doumont, J. (2002). Magical numbers: the seven-plus-minus-two myth. *IEEE Transactions on Professional Communication*, 45(2):123–127.

[Drescher, 1991] Drescher, G. (1991). *Made-up minds: a constructivist approach to artificial intelligence.* MIT Press.

[Gerkey and Mataric, 2004] Gerkey, B. P. and Mataric, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *Intl. J. of Robotics Research*, 23(9):939–954.

[Goldberg, 2002] Goldberg, D. (2002). *The Design of Innovation (Genetic Algorithms and Evolutionary Computation).* Springer.

[Gritti et al., 2007] Gritti, M., Broxvall, M., and Saffiotti, A. (2007). Reactive self-configuration of an ecology of robots. In *Proc. of the ICRA-07 Workshop on Network Robot Systems*, Roma, Italy.

[Gureckis and Love, 2004] Gureckis, T. M. and Love, B. C. (2004). Common mechanisms in infant and adult category learning. *Infancy*, 5(2):173–198.

[Hansen et al., 1997] Hansen, E. A., Zilberstein, S., and Danilchenko, V. A. (1997). Anytime heuristic search: first results. Technical Report 97-50, Computer Science Department, University of Massachusetts.

[Hart et al., 2004] Hart, S., Grupen, R., and Jensen, D. (2004). A relational representation for generalized knowledge in robotic tasks. Technical Report 04-101, Computer Science Department, University of Massachusetts Amherst.

[Huang and van de Panne, 1996] Huang, P. S. and van de Panne, M. (1996). A planning algorithm for dynamic motions. In *Computer Animation and Simulation '96*, pages 169–182.

[Huntsberger et al., 2003] Huntsberger, T., Pirjanian, P., Trebi-Ollennu, A., Nayar, H. D., Aghazarian, H., Ganino, A., Garrett, M., Joshi, S., and Schenker, P. (2003). Campout: a control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 33(5):550–559.

[Jones et al., 2006] Jones, E., Browning, B., Dias, M. B., Argall, B., Veloso, M., and Stentz, A. (2006). Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 570 – 575, Orlando, FL.

[Konda and Tsitsiklis, 1999] Konda, V. R. and Tsitsiklis, J. N. (1999). Actor-critic algorithms. In *Advances in Neural Information Processing Systems 12, NIPS Conference*, pages 1008–1014, Denver, Colorado, USA.

[Konidaris and Barto, 2006a] Konidaris, G. D. and Barto, A. G. (2006a). An adaptive robot motivational system. In *Animals to Animats 9: Proceedings of the 9th International Conference on Simulation of Adaptive Behavior*, CNR, Roma, Italy.

[Konidaris and Barto, 2006b] Konidaris, G. D. and Barto, A. G. (2006b). Building portable options: skill transfer in reinforcement learning. Technical Report UM-CS-2006-17, University of Massachusetts Department of Computer Science Technical Report.

[Konishi and Fujii, 2004] Konishi, Y. and Fujii, R. H. (2004). Incremental learning of temporal sequences using state memory and a resource allocating network. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2889– 2893.

[LeBlanc and Saffiotti, 2008] LeBlanc, . K. and Saffiotti, A. (2008). Cooperative anchoring in heterogeneous multi-robot systems. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Pasadena, CA.

[Levner et al., 2006] Levner, I., Kovarsky, A., and Zhang, H. (2006). Heuristic search for coordinating robot agents in adversarial domains. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 563–569, Orlando, FL.

[Lundh et al., 2007] Lundh, R., Karlsson, L., and Saffiotti, A. (2007). Plan-based configuration of an ecology of robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Roma, Italy.

[Lyons and Arbib, 1989] Lyons, D. M. and Arbib, M. A. (1989). A formal model of computation for sensory-based robotics. *IEEE Transactions on Robotics and Automation*, 5(3):280–293.

[Mahadevan and Connell, 1991] Mahadevan, S. and Connell, J. (1991). Automatic programming of behavior-based robots using reinforcement learning. In *National Conference on Artificial Intelligence*, pages 768–773.

[Newell and Simon, 1963] Newell, A. and Simon, H. A. (1963). Gps, a program that simulates human thought. In Feigenbaum, E. A. and Feldman, J., editors, *Computers and Thought*. New York: McGraw-Hill.

[Oztop et al., 2004] Oztop, E., Bradley, N. S., and Arbib, M. A. (2004). Infant grasp learning: a computational model. *Experimental Brain Research*, 158(4):480–503.

[Parker et al., 2004] Parker, L. E., Kannan, B., Tang, F., and Bailey, M. (2004). Tightly-coupled navigation assistance in heterogeneous multi-robot teams. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, Sendai, Japan.

[Parker and Tang, 2006] Parker, L. E. and Tang, F. (2006). Building multi-robot coalitions through automated task solution synthesis. *Proceedings of the IEEE, special issue on Multi-Robot Systems*, 94(7):1289–1305.

[Piaget, 1952] Piaget, J. (1952). *The psychology of intelligence*. London: Routledge and Kegan Paul LTD.

[Piaget, 1963] Piaget, J. (1963). *The origins of intelligence in children*. New York: The Norton Library, W. W. Norton and Company Inc.

[Piaget, 1981] Piaget, J. (1981). *Intelligence and affectivity: their relationship during child development*. Annual Reviews Inc., Palo Alto, California, USA.

[Platt et al., 2005] Platt, R., Fagg, A., and Grupen, R. (2005). Reusing schematic grasping policies. In *Proceedings of IEEE-RAS International Conference on Humanoid Robots*, Tsukuba, Japan.

[Platt et al., 2006] Platt, R., Grupen, R., and Fagg, A. (2006). Improving grasp skills using schema structured learning. In *Proceedings of International Conference on Development and Learning*, Bloomington, Indiana.

[Robins and McCallum, 1999] Robins, A. and McCallum, A. (1999). The consolidation of learning during sleep: comparing the pseudorehearsal and unlearning accounts. *Neural Networks*, 12(7-8):1191–1206.

[Saffiotti and Broxvall, 2005] Saffiotti, A. and Broxvall, M. (2005). Peis ecologies: Ambient intelligence meets autonomous robotics. In *Proc. of the sOc-EUSAI conference on Smart Objects and Ambient Intelligence*, Grenoble, France.

[Saffiotti et al., 2008] Saffiotti, A., Broxvall, M., Gritti, M., LeBlanc, K., Lundh, R., Rashid, J., Seo, B., and Cho, Y. (2008). The peis-ecology project: vision and results. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Nice, France.

[Schlechter and Henrich, 2006] Schlechter, A. and Henrich, D. (2006). Discontinuity detection for force-based manipulation. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1378–1783.

[Shan and Tan, 2006] Shan, X. and Tan, J. (2006). Multi-robot coordination for elusive target interception aided by sensor networks. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5540–5545.

[Shehory and Kraus, 1999] Shehory, O. and Kraus, S. (1999). Feasible formation of coalitions among autonomous agents in non-super-additive environments. *Computational Intelligence*, 15(3).

[Sheng et al., 2006] Sheng, W., Yang, Q., Tan, J., and Xi, N. (2006). Distributed multi-robot coordination in area exploration. *Robotics and Autonomous Systems*, 54(12):945–955.

[Simonin et al., 2005] Simonin, E., Diard, J., and Bessiere, P. (2005). Learning bayesian models of sensorimotor interaction: from random exploration toward the discovery of new behaviors. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1226–1231.

[Şimşek and Barto, 2006] Şimşek, Ö. and Barto, A. G. (2006). An intrinsic reward mechanism for efficient exploration. In *Proceedings of the Twenty-Third International Conference on Machine Learning.*

[Stroupe et al., 2005] Stroupe, A., Huntsberger, T., Okon, A., Aghazarian, H., and Robinson, M. (2005). Behavior-based multi-robot collaboration for autonomous construction tasks. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1495-1500.

[Tang, 2006] Tang, F. (2006). *ASyMTRe: Building coalitions for heterogeneous multi-robot teams.* PhD thesis, The Department of Computer Science, The University of Tennessee.

[Tang and Parker, 2005a] Tang, F. and Parker, L. E. (2005a). ASyMTRe: automated synthesis of multi-robot task solutions through software reconfiguration. In *Proceedings of IEEE International Conference on Robotics and Automation(ICRA).*

[Tang and Parker, 2005b] Tang, F. and Parker, L. E. (2005b). Coalescing multi-robot teams through ASyMTRe: A formal analysis. In *Proceedings of IEEE International Conference on Advanced Robotics (ICAR).*

[Tang and Parker, 2005c] Tang, F. and Parker, L. E. (2005c). Distributed multi-robot coalitions through ASyMTRe-D. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

[Tang and Parker, 2006] Tang, F. and Parker, L. E. (2006). Layering coalition formation with task allocation. In *Proceedings of the AAAI Workshop: Auction Mechanisms for Robot Coordination.*

[Tang and Parker, 2008] Tang, Y. and Parker, L. E. (2008). Towards schema-based, constructivist robot learning: Validating an evolutionary search algorithm for schema chunking. In *Proceedings of the IEEE Conference on Robotics and Automation.*

[Taylor and Taylor, 2000] Taylor, N. R. and Taylor, J. G. (2000). Is there more to TSSG than associative chaining (chunking and all that)? In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*, pages 217–222.

[Tedrake et al., 2005] Tedrake, R., Zhang, T. W., and Seung, H. S. (2005). Learning to walk in 20 minutes. In *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, Yale University, New Haven, CT.

[Wang and de Silva, 2006] Wang, Y. and de Silva, C. (2006). Cooperative transportation by multiple robots with machine learning. In *Proc. of the IEEE Congress on Evolutionary Computation (CEC)*, pages 3050–3056.

[Wang and de Silva, 2008] Wang, Y. and de Silva, C. W. (2008). A machine-learning approach to multi-robot coordination. *Engineering Applications of Artificial Intelligence*, 21(3):470–484.

[Werbos, 1997] Werbos, P. J. (1997). A hybrid hierarchical neural-AI model of mammal-like intelligence. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 'Computational Cybernetics and Simulation'*, pages 1442–1444, Orlando, FL, USA.

Vita

Yifan Tang joined the Department of Computer Science at the University of Tennessee-Knoxville (UTK) as Graduate Student in August 2002. She received the M.S. degree in computer science from UTK. She has also received a Diploma Kauffrau (equivalent to MBA) from the Tuebingen University in Germany and an M.S. degree in computer science from Minnesota State University, Mankato.

She has been working with Dr. Parker since she joined the Distributed Intelligent Laboratory at UTK in 2003. She worked on the Software for Distributed Robotics project funded by DARPA from 2003 to 2005. Her dissertation research is to develop a continuous robot learning architecture inspired by human constructivist learning. Her research interests include machine learning, data mining, and educational robotics. She is a student member of IEEE and ACM.