Heuristics and Genetic Algorithms for Adaptive Deployments in Large-scale, Real-time Systems

James Edmondson and **Aniruddha Gokhale** and **Sandeep Neema**
Dept of EECS, Vanderbilt University
Nashville, TN 37212, USA
{james.r.edmondson,a.gokhale,sandeep.neema}@vanderbilt.edu

Abstract

Distributed, real-time and embedded (DRE) applications are often deployed into mission-critical scenarios where money or lives are at stake. Examples of these types of scenarios include search-and-rescue missions, shipboard computing, satellite infrastructure, power grids, and air traffic control. Once deployed, these DRE applications must service important requests continuously and perpetually, but over time, the environment may change. Hardware may fail or degrade in performance. Computers may move to other locations where latency is worse between important parts of the DRE application. Despite these changes in the environment, the DRE application needs to adapt and continue to respond and operate. In the past, developers or system administrators were required to manually update or reboot the DRE application, analyze or guess the best remaining configurations, and move important processing to the best hardware available. Each of these steps are prone to human error that may result in unacceptable configurations of the DRE application. In this paper, we discuss ongoing work in the context of redeployment of adaptive, mission-critical DRE applications via new heuristics and genetic algorithms that approximate the subgraph isomorphic problem, a known NP complete problem, and middleware and tools that use the results of these heuristics to automate the redeployment process. We provide results that show some of the scenarios that result in perfect deployment approximations, and we also motivate future work to address the blind spots in our approximation techniques.

1 Introduction

Distributed, real-time and embedded (DRE) systems are often characterized by stringent quality-of-service needs despite scarce resources (*e.g.*, CPU, memory, or network capacity). Of the types of DRE systems, none are harder to meet quality-of-service requirements than continuous, adaptive large-scale systems—which require online solutions to meet their needs. In this paper, it is within this context that we motivate our work in distributed middleware development.

We are creating a suite of open-source tools called the Multi-Agent Distributed Adaptive Resource Allocation (MADARA) suite, which currently includes a high performance knowledge and reasoning engine, and an automated testing and deployment toolset for large-scale DRE systems. We focus our description on our ongoing work with the deployment engine to accomodate online, continuously optimized redeployments according to a high-level application data-flow.

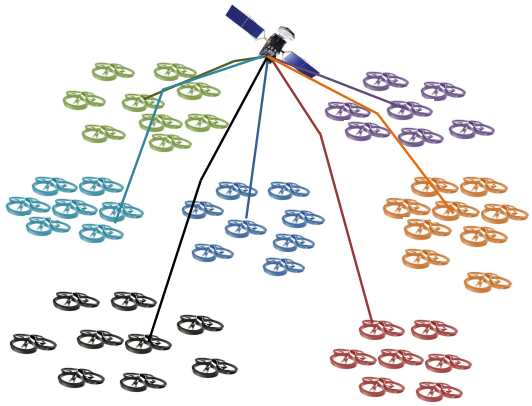

Figure 1: Satellite-connected real-time system deployment consisting of flying drones in a search and rescue mission

In this section, we motivate our work with the scenario shown in Figure 1. In this situation, a search-and-rescue operation is underway to detect and aid survivors of an earthquake or other catastrophic event. Thousands of airborne drones are deployed into a massive disaster area. The drones are fitted with cameras and sensors to aid in their search for stranded or trapped survivors, but drone operators have to balance data and power concerns with how they deploy these mission-critical drones.

Disaster areas, like those that occur after earthquakes, rarely have full network connectivity (*e.g.*, internet), charging stations, and power. Additionally, gas leaks or even radiation leaks (*e.g.*, recently after the earthquake and tsunami that disabled the Fukushima reactors) may make the area unapproachable to the drone operators or rescue personnel and may also disrupt long-range communications (*e.g.*, radio links and line-of-sight based communications), causing blackout zones in the search area without the aid of autonomous agents like the drones.

Consequently, human-based drone operators and analyzers of the drone data are likely to be restricted to a satellite link or a set number of low-bandwidth radio links into the disaster zone, and using this link is constrained and expensive—in terms of power usage for the drones. Drone operators will be unable to receive direct feeds from each of the thousands of drones in the area, and are going to need to receive video or sensor data from a small subset of drones—possibly from only a handful, if the satellite or radio bandwidth available is in the kilobytes per second range.

In order to maximize the time that these drones can service the disaster area, the operators create a data workflow that describes the way that video and sensor data will be collected and how the drones will need to self-organize in the field so that one of them out of a large group can communicate with the operators, and the number of such special drones that serve as collection points will be dependent on the available bandwidth and estimated power usage of the communication mechanism available between human operators and the disaster area. An example of such a workflow is shown in Figure 2.

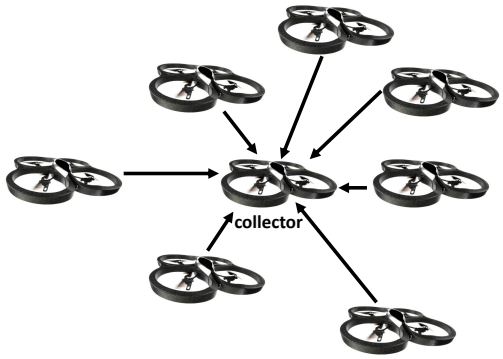

Figure 2: Example data workflow for one of the dynamically formed drone subgraphs

A short path between the collector drone and the drones it services is important because the longer the path between them, the more likely that data resends are required and the longer that communication takes, which again is the highest drain on the drone's battery-life. If we can minimize the latencies that exist in deployments like Figure 2, which in turn compromise the subgraphs of the thousands of drones deployed in the configuration shown in Figure 1, then we can increase the longevity of the drones and potentially find more survivors with less downtime due to returning the drones to a demarcation point for recharging and also allow for the drones to collect data locally and only sending data, images or video that has a high likelihood of locating a survivor. An initial deployment is unlikely to remain the optimal one because drones may move away from their group when they find structures that may have humans in them, hear sounds or have sensors for bodyheat, or even to avoid obstacles, which means that any deployment will have to be elastic to change and must be solved continuously and online.

The real problem with creating an online, continuous solution to this scenario is that this is a provably NP complete problem. Specifically, this motivating scenario is a type of subgraph isomorphism problem (Garey and Johnson 1979) where two graphs are given as input and one must determine if one of the graphs (like the one in Figure 2) is contained in the larger one (in this case, the overall cloud of drones that are available for the subgraph as shown in Figure 1 but at a larger scale of thousands). When you add multiple subgraphs, as we do in this scenario, it compounds the intractibility. This is also an optimization problem because out of all the possible subgraphs, we are interested in finding the best one (the one with the lowest total latency—which directly reflect radio transmission times and power usage for the drones) that fits with all of the other data workflows in the overall network.

In this paper, we discuss our work in approximating this problem with heuristics and genetic algorithms. We validate these approaches with certain types of subgraphs in large deployments of thousands of drones or components.

In Section 2, we outline related work in approximating these types of NP complete problems. Section 3 defines some of our own solutions, including heuristics and genetic algorithms. Section 4 presents experiments and results of our techniques in approximating not only the motivating scenario but also other highly-interconnected types of subgraphs. Section 5 discusses remaining challenges in our open problem and status of the toolset. Finally, Section 6 discusses the contributions of the paper as well as future work.

2 Related Work

In this section, we investigate related work in subgraph isomorphic solutions and techniques for approximating NP complete problems.

2.1 Approximations for Subgraph Isomorphic Problem

The LeRP algorithm (DePiero and Krout 2003) forms a node-to-node mapping from a neighborhood of nodes and can be computed with polynomial effort, producing a result for a 50-100 node subgraph within half a second. It is not apparent whether or not this approach can handle disruptions in the local network that result in inexact subgraph matches (*e.g.* or when drones are allowed to route traffic through others close by, which we've found is relavant to our motivating scenario. Still, we hope to look into this heuristic in future analyses.

Another framework was recently presented (Zampelli, Deville, and Dupont 2005) that allows for constraints mapped onto a pattern intended for subgraph isomorphic matching. The main problem with this approach is the time that it takes. A target graph of 20-200 can take between 8 to 36 minutes and still has double digit percentages of unsolved subgraphs. This is unacceptable for an online, mission critical application such as that defined in the motivating scenario.

2.2 Constraint Satisfaction Problem Solving

Haldik et. al. (Hladik et al. 2008) presented a constraint programming technique to solve static allocation problems in real-time tasks. Cucu-Grosjean et. al. (Cucu-Grosjean and Buffet 2009) proposed two approaches to addressing real-time periodic scheduling on heterogeneous platforms (a CSP problem). Despite being faster than traditional CSP solvers, both techniques require dozens to hundreds of seconds to solve even small number of constraints (and our motivating scenario requires thousands).

White et. al. (White et al. 2008) recently scaled a CSP solver to work with 5,000 features in a software product line. The time required for doing this ranged from 50 seconds in an incomplete bounded worst case to 170 seconds to find an optimal configuration. Although this is a promising result, it is still inadequate for continuous online systems.

2.3 Research in Genetic Algorithms for CSPs

Whereas CSP solving typically involves backtracking through potential matches, genetic algorithms are a type of local search that tries to approximate an optimal match through mutations, fitness functions, and crossbreeding best candidates according to the fitness criteria.

Heward et. al. (Heward et al. 2011) recently used genetic algorithms to optimize configurations of monitors in a web services application. Unfortunately, this method is unsuitable for our use cases as it requires roughly an hour to compute an approximated good configuration.

Wieczorek et. al. (Wieczorek, Prodan, and Fahringer 2005) used a genetic algorithm to schedule scientific workflows in Grid environments, but their mutation-based scheme similarly required at least hundreds of seconds (some of their tests showed requirements of tens of thousands of seconds—several hours).

Other implementers have used combinations of genetic algorithms and neural networks (Javadi, Farmani, and Tan 2005) and even knowledge and reasoning (Hu and Yang 2004) to converge to optimal solutions, but each of these concentrate on offline or human-interactive solutions, and are not suitable to real-time problem solving because they require many minutes or hours to approximate a solution, and our motivating scenario is an evolving, real-time problem.

2.4 Research in Heuristics for CSPs

Heuristics approximate good solutions and often serve as guides for local search techniques like genetic algorithms, simulated annealing, or backtracking and depth-first searches. Some researchers use these heuristics to directly approximate scheduling (Heward et al. 2011) in grids and workflow solutions (Cucinotta and Anastasi 2011) for real-time solutions. The latter is of special interest to us as the heuristic approximates a constraint problem involving a set of workflows within milliseconds. However, this solution was demonstrated on only five hosts and not thousands, and it is not readily apparent how to migrate our motivating scenario to the heuristic defined in either of the related papers.

The heuristic-based anytime A* search (Hansen and Zhou 2007) is similar in some ways to our approach to genetic algorithms for this motivating scenario in that both solutions may be stopped at any time and a solution is presented to the user (though it may not necessarily be optimal). The heuristics offered in this paper may be used with an A* search, and may be the subject of future work.

We are unaware of any deployment frameworks, other than ours, that approximate this kind of problem in real-time situations.

Algorithm 1 CID Heuristic

```
 1: for all i ∈ deployment do
 2:   if degree (i) > 0 then
 3:     solution[i] ← best_candidate (utilities[degree(i)])
 4:   end if
 5: end for
 6: for all i ∈ deployment do
 7:   if degree (i) > 0 then
 8:     for j ∈ connections(deployment, i) ∧ j ∉ solved(solution) do
 9:       solution[j] ← best_candidate (latencies[i])
10:     end for
11:   end if
12: end for
13: for all i ∈ deployment ∧ i ∉ solved(solution) do
14:   solution[i] ← best_candidate (utilities[size])
15: end for
```

3 Solution Approach

In this section, we discuss the heuristics and genetic algorithms that we are currently using to approximate the problem discussed in the motivating scenario.

3.1 Heuristics

The first solution approach we discuss in this paper is the Comparison-based Iteration By Degree (CID) Heuristic, which focuses on solving deployments by the connectivity (which we refer to as the degree) of each node in the data workflow. This particular heuristic (shown in Algorithm 1) does extremely well with the motivating scenario because the subgraphs are disjoint—each collection group is isolated from each other. The graphs do not have dependencies between each other and this makes the degree information extremely useful and accurate for the motivating scenario.

The first phase of Algorithm 1 (lines 1-5), iterates through a deployment with candidates sorted in descending order by outgoing degree. The best candidates are placed in the deployment, and the heuristic then solves for those candidates that had edges to already placed entities in the deployment (lines 6-12). The final phase of the heuristic solves for any entity in the deployment that the user has defined as isolated or unnecessary for optimization (lines 13-15).

The second heuristic discussed here is called the Blind Comparison-based Iteration By Degree And Path Latency (BCID) Heuristic and is shown in Algorithm 2. This heuristic was developed after the initial CID heuristic to attempt

Algorithm 2 Blind CID

```
1: for all i ∈ deployment do
2:   if degree (i) > 0 then
3:     solution[i] ← best_candidate (utilities[degree])
4:   end if
5: end for
6: for all i ∈ deployment do
7:   if degree (i) > 0 then
8:     for j ∈ connections(deployment, i) ∧ j ∉ solved(solution) do
9:       solution[j] ← best_candidate (utilities[size])
10:    end for
11:  end if
12: end for
13: for all i ∈ deployment ∧ i ∉ solved(solution) do
14:   solution[i] ← best_candidate (utilities[size])
15: end for
```

a looser approximation by only using the nodes with the best overall utility. This heuristic has an additional benefit of not requiring the individual latency tables for all processes, which means memory requirements for this heuristic are only O(E), where E is the size of the drones available for deployment in the environment rather than $O(E^2)$ for Algorithm 1. In terms of our motivating scenario, this is the difference between requiring drones to maintain 40KB of latencies versus 400MB of latencies for a 10,000 drone environment. Consequently, even if this heuristic is less precise for many deployments, it has the important benefit of putting a smaller memory constraint on our passive redeployment logic.

Algorithm 3 Blind GA

```
1: mutations ← min + rand() % (max - min)
2: new ← solution
3: orig_utility ← utility(new)
4: while maxtime has not elapsed do
5:   c1 ← rand() % size
6:   c2 ← rand() % size
7:   while c1 ≡ c2 do
8:     c2 ← rand() % size
9:   end while
10:  swap(solution[c1], solution[c2])
11: end while
12: if utility(new) < orig_utility then
13:   return new
14: else
15:   return solution
16: end if
```

To complement these heuristics, we developed a pair of genetic algorithms to improve on the solutions and attempt to reduce the overall latency in the set of subgraphs. These genetic algorithms are shown in Algorithms 3 and 4. The first algorithm blindly chooses chromosomes of the solution to change with candidates from the cloud of drones or other pieces of the subgraphs.

The second algorithm (Algorithm 4) targets chromosomes of the solution with the largest degrees (the connectedness of the node) in the subgraph. It swaps these with other larger degrees more often than it does with random parts of the subgraph, as the blind genetic algorithm does. This results in more targeted evolution of the solution, which we've noticed is better for approximating workflows with few dependencies between subgraphs but may perform worse than Algorithm 3 on highly interdependent subgraphs, which we'll see examples of in Section 4.

Algorithm 4 Guided GA

```
1: mutations ← min + rand() % (max - min)
2: new ← solution
3: orig_utility ← utility(new)
4: while maxtime has not elapsed do
5:   if rand() % 5 < 4 then
6:     c1 ← random_degreed_node (deployment)
7:     c2 ← location(solution[good_candidate(utilities)])
8:     while c1 ≡ c2 do
9:       c2 ← location(solution[good_candidate(utilities)])
10:    end while
11:  else
12:    c1 ← rand() % size
13:    c2 ← rand() % size
14:    while c1 ≡ c2 do
15:      c2 ← rand() % size
16:    end while
17:  end if
18: end while
19: if utility(new) < orig_utility then
20:   return new
21: else
22:   return solution
23: end if
```

Both of these genetic algorithms are allowed to make alterations for a certain amount of time. In our experiments, we set this time to one second, but the framework is flexible in allowing users to specify subsecond times or large numbers of seconds.

4 Experiments

In this section, we analyze the performance of and utility produced by the algorithms and heuristics detailed in Section 3. All experiments were conducted on an Intel Core2 Duo clocked at 2.53 GHz and 4 GB of RAM running Windows 7 32-bit operating system. C++ Code was compiled in Visual Studio 2008 under optimized Release. All experimental code and configuration information can be found online on the MADARA project site at madara.googlecode.com.

Each of the genetic algorithm-based approaches were allowed to run for 5 seconds, resulting in thousands of chromosome mutations. This amount of time may not be acceptable for all systems, and the MADARA suite of tools do allow for setting an arbitrary time as low as milliseconds to

attempt mutations. The CID and BCID heuristics each require less than 20 milliseconds of runtime to approximate a 10,000 drone deployment.

For all experiments, we generate one truly optimal solution within the network that has 500 us latency between the important links. Every other possible connection has latency in the 501us to 32ms range. If the heuristic or algorithm finds the optimal deployment, the resulting system slowdown is 1. Anything else will be greater than 1. System slowdown factor is calculated by dividing the total latency of each link in the approximation by the optimal total latency.

4.1 Disjoint Subgraphs

The motivating scenario for this paper consisted of collection drones that aggregated data from their local drone groups and then communicated with human operators over a constrained satellite or radio link. To mimic this scenario, we generated a uniform distribution of latencies between 1,000 to 10,000 nodes in a drone deployment.

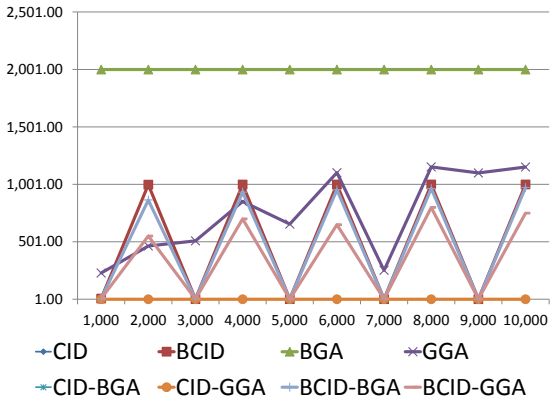

Figure 3: System Slowdown From Approximations Generated for Motivating Scenario

Figure 3 shows the system slowdown experienced in each of the solutions produced by the heuristics and genetic algorithms. The y-axis denotes the slowdown factor of the resulting system, and the x-axis indicates the size of the deployment. All algorithms that use the CID heuristic (CID, CID-BGA, and CID-GGA in the figures) are able to produce the optimal solution in each of the test cases for 1,000 to 10,000 nodes and do so within milliseconds. The CID heuristic is so powerful in this scenario because degree information directly reflects the subgraphs that need to be solved, and many scenarios involving optimization of backup servers, brokering services, and data aggregation or broadcasting do well with the CID heuristic.

4.2 Highly Interdependent Scenarios

Though the CID heuristic solves the motivating scenario in a fraction of a second, we knew it would not solve all of the possible subgraphs because the general problem is NP Complete. In this section, we look at the results of our algorithms and heuristics in approximating hierarchical, complete trees consisting of 1,000 to 10,000 nodes. As with the motivating scenario, we combined each of the heuristics and algorithms and ran them 10 times each for indicated sizes.

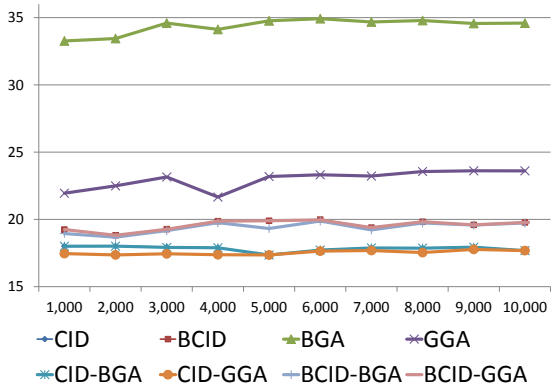

Figure 4: System Slowdown of a 3-depth Tree

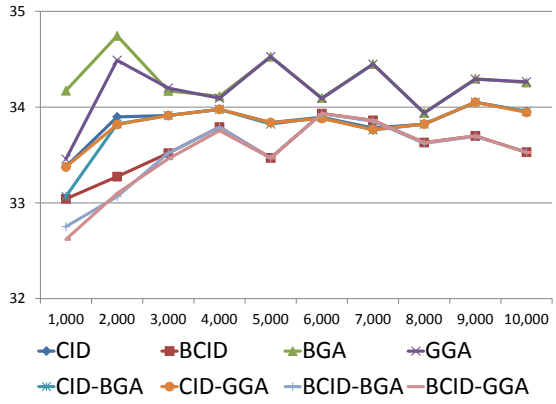

Figure 5: System Slowdown of a 3-way Tree

Figure 4 shows the system slowdown experienced when approximating a 3 depth tree where drone 0 collects data from drone 1-4, and drone 1-4 each collect data from a fourth of the deployment. All approximations are more than 15 times worse than the optimal.

Figure 5 shows the system slowdown that results when approximating a 3-way tree, where a root drone collects data from 3 other drones, which in turn collect data from 3 other drones and send the important data to the root. The slowdown factor when approximating these is even worse because every level of the tree has the same degree except for the leaves, and the arbitrary ordering of the heuristics according to degree results in ties, but the resolution order of these should be important.

These latter hierarchical scenarios show the problem with using these heuristics to approximate general cases of interdependent deployments. A full communication originating from all leaf nodes to the root node at drone 0 would be 15-

35 times worse than the optimal. The CID and BCID heuristics do decently well in the 3-depth scenario, and the genetic algorithms do improve the results, but we have a lot of room for improvement here. From these experiments, we believe that an excellent vector for future work will be a path-length- and degree-based heuristic that is then used to seed a specialized genetic algorithm.

5 Challenges

In this section, we describe remaining challenges for addressing continuous redeployment of distributed, real-time and embedded (DRE) systems. We can divide these challenges into three main categories: better approximation, custom constraints, and semantics in continuous redeployment.

5.1 Better Approximation

From the results shown in Section 4, we know that Algorithms 1 and 2 work well for approximating user-defined data workflows that do not have a lot of interdependencies between subgraphs. However, these heuristics and the genetic algorithms we have paired them with are unable to find the optimal deployment of elaborate workflows like m-way trees.

We believe that degree information may still be the key to unlocking the optimal solutions in such high interdependent graphs, and we plan on trying other heuristics and genetic algorithms that use degree and path information to converge more quickly to optimum redeployment solutions.

5.2 Custom Constraints

We have ongoing work with overlaying custom constraints onto edges and nodes in the data workflow to provide better solutions for DRE system developers. Example constraints include CPU and memory requirements for each node, latency requirements along path lengths and other such metrics. Some of these constraints may force additional complexity onto the solution approaches, but they may also relax the optimization requirements (*.e.g.*, a requirement that the latency along a link is less than 500 us is easier to solve than requiring that such a link is optimal).

5.3 Continuous Semantics of Redeployments

When redeploying a running distributed application like the one in the motivating scenario, we shouldn't have to transfer databases or other state information to the new entity that assumes the role. However, many DRE systems have databases, files, logic, or other such side effects that must be redeployed. We are working on tools to address these needs.

6 Conclusions

In this paper, we have presented a subset of the MADARA tool suite that approximates optimal deployments of distributed, real-time and embedded (DRE) applications and provides continuous redeployment infrastructure for high-impact, mission-critical scenarios. We have motivated our problem with an example mission-critical application and presented experiments that validated our heuristics and algorithms within this scenario. We also outlined experimental results that show blind spots in the heuristics and genetic algorithms when applied to hierarchical deployments and discussed our plans to develop more robust heuristics-based approaches that better approximate the general NP-complete subgraph isomorphic problem as it correlates to real-time deployments.

References

Cucinotta, T., and Anastasi, G. 2011. A heuristic for optimum allocation of real-time service workflows. In *Service Oriented Computing and Applications, 2011. SOCA '11. International Conference on*, 169–172.

Cucu-Grosjean, L., and Buffet, O. 2009. Global multiprocessor real-time scheduling as a constraint satisfaction problem. In *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*, 42–49.

DePiero, F., and Krout, D. 2003. An algorithm using length r paths to approximate subgraph isomorphism. *PATTERN* 24:33–46.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Hansen, E. A., and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research (JAIR* 28:267–297.

Heward, G.; Han, J.; Schneider, J.-G.; and Versteeg, S. 2011. Run-time management and optimization of web service monitoring systems. In *Service Oriented Computing and Applications, 2011. SOCA '11. International Conference on*, 294–299.

Hladik, P.-E.; Cambazard, H.; Daplanche, A.-M.; and Jussien, N. 2008. Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software* 81(1):132–149.

Hu, Y., and Yang, S. 2004. A knowledge based genetic algorithm for path planning of a mobile robot. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, 4350 – 4355 Vol.5.

Javadi, A.; Farmani, R.; and Tan, T. 2005. A hybrid intelligent genetic algorithm. *Advanced Engineering Informatics* 19(4):255 – 262.

White, J.; Schmidt, D.; Benavides, D.; Trinidad, P.; and Ruiz-Cortes, A. 2008. Automated diagnosis of product-line configuration errors in feature models. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, 225–234.

Wieczorek, M.; Prodan, R.; and Fahringer, T. 2005. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Rec.* 34:56–62.

Zampelli, S.; Deville, Y.; and Dupont, P. 2005. Approximate constrained subgraph matching. In van Beek, P., ed., *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 832–836.