Cooperative search for optimizing pipeline operations

T. Mora, A.B. Sesay
Department of Electrical and Computer Engineering
University of Calgary
Calgary, Canada
{temora,sesay}@ucalgary.ca

J. Denzinger
Department of Computer Science
University of Calgary
Calgary, Canada
denzinge@cpsc.ucalgary.ca

H. Golshan, G. Poissant, C. Konecnik
TransCanada PipeLines Limited
{hossein_golshan,gene_poissant, cameron_konecnik}@transcanada.com

ABSTRACT

We present an application of a multi-agent cooperative search approach to the problem of optimizing gas pipeline operations, i.e. finding control parameters for a gas transmission network that result in a low usage of energy to make the required gas deliveries. Our cooperative search approach improves on the pure competition of search agents by having them exchange good solutions from time to time that both are integrated into the search state of the agents and used to improve the search control of the agents. Our experimental evaluation with real problem instances from TransCanada show that our system meets TransCanada's time requirements and reliably outperforms the interactive method that is the current state-of-the-art by creating solutions that require more than 10 percent less energy.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; I.2.8 [**Problem Solving, Control Methods and Search**]: Heuristic methods; J.7 [**Computers in other systems**]: Industrial control

General Terms

Algorithms, Experimentation

Keywords

Cooperative search, pipeline operations, optimization

1. INTRODUCTION

Cooperative problem solving by a group of agents was an early focus of research in multi-agent systems. The hopes of such research was to speed-up the efficiency of problem solving and perhaps even to achieve synergetic effects. This was especially true for problems that require as solution method some kind of search, a solution method that usually employs heuristical controls and as a consequence performs many computation steps that in hindsight, after having found the solution, are revealed as unnecessary. Multi-agent approaches to achieve cooperative problem solving using search have been shown to achieve synergetic effects (see [4], [7], [13]) and also to deal with additional problems like naturally distributed search problems (see [15]) and many distribution approaches to different search paradigms are best described using multi-agent systems (see [5]) even if originally they were not presented this way (see, for example, [1], [8], or [10]).

But despite the reported successes for well-known search problems from literature, the use of multi-agent search approaches to solve difficult industrial and business problems has been rather limited. A notable exception is [11]. One of the reasons for this lack of application might have been that the advances in computer hardware have been so rapid that waiting a little bit was enough to get the problems into reach. Solving an instance of a search problem that required too much run time at one point in time was easily possible one or two hardware generations later, which often was only a few months later. And this is usually less time than would have been needed to develop a new system utilizing multi-agent search with agents that use their own processors or computers.

Unfortunately, in recent years the performance improvements achieved by new hardware generations, while still growing steadily if we define performance by, for example, basic computations per second, is now achieved by multiple core and/or multiple processor machines that produce the increases essentially due to parallel processing. While some applications are relatively easy to modify towards being solved by several processors, most search algorithms constantly re-evaluate their possible steps after each step taken, so that simple distribution concepts do not result in improvements over the sequential algorithms (only in more unnecessary steps taken). Multi-agent cooperation concepts are needed to use the multi-core/multi-processor technology for search!

An application area where the complexity of problem instances outgrew even the improvements in computer hardware is the optimization of pipeline operations. Finding control parameters to operate compressor/pumping stations and other devices in transportation networks that achieve the required deliveries out of the output stations of the network or subnetwork with a minimal consumption of energy and within the various system constraints is a difficult problem. The sets of equations that describe and model the system are too complex to allow for a direct solution, so that every solution candidate for a problem instance has to be evaluated using a simulation of the pipeline network, which is a rather costly step with regard to the computing effort involved. Additionally, the fact that some parame-

Cite as: Cooperative search for optimizing pipeline operations, T. Mora, J. Denzinger, H. Golshan, et al., *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)-Industry and Applications Track*, Berger, Burg, Nishiyama (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 115-122.

ters are real values adds the need to deal with continuous optimization, while there is nevertheless a strong discrete optimization component. As a consequence, there is a lot of heuristical knowledge around this application that often has as much potential to confuse as to help.

In this paper, we present the results of applying the TECHS concept for cooperative search (see [7]) to the optimization of the operation of some of TransCanada's subnetworks of gas pipelines. More precisely, we use a homogeneous variant of TECHS with agents that perform a set-based search using particle swarms and periodically exchange selected information. Our IOPO system, which instantiates the TECHS concept for our application, was able to produce solutions for complex networks that resulted in more than 10 percent less cost than the current interactive method commercially available taking approximately the same time (but being fully automated). Our experiments also reveal that the multi-processor/multi-core technology and current operating systems still do not perfectly support multi-agent search approaches.

This paper is organized as follows. After this introduction, in Section 2 we present the industrial application problem of optimizing gas pipeline operations. In Section 3, we provide some general notations about multi-agent search systems and present the TECHS concept for cooperative search. Section 4 presents TECHS' instantiation to our application area. Section 5 reports on the experiments we performed with pipeline networks from TransCanada, comparing our approach to a commercial method and another approach from literature, and analyzes the contribution of multi-agent concepts and a multi-processor/multi-core hardware architecture. Finally, in Section 6, we conclude with some lessons learned and possible future work.

2. THE PROBLEM: OPTIMIZATION OF GAS PIPELINE OPERATIONS

The problem of interest to TransCanada is the optimization of the operation of their natural gas pipeline transmission system. A *pipeline transmission system* is represented by a complex network that may consist of hundreds of nodes, devices and other equipment to control. Natural gas is generally received from receipt points along the pipeline network and delivered to sales stations at specified flows and pressures. Between these points pressure drop occurs due to gas expansion, friction loss, changes in elevation and changes in temperature ([9]). Compression is required to overcome the pressure losses that occur over the length of the pipeline. Adding compressor stations at intervals along the pipeline network is one of the solutions used to achieve and maintain the required pressure. Natural gas-fired turbine engines are the most common drivers for compressors on TransCanada's gas transmission pipelines in Canada. Gas turbines spin centrifugal compressors and compress the gas up to a hundred times normal atmospheric pressure to move the gas. Reduction of the energy used in pipeline operations not only has a tremendous economical impact but also quite an environmental one. More efficient operation of compressor stations results in less greenhouse gas emissions being dissipated to the atmosphere.

Operation of a natural gas pipeline network implies the selection of all operational settings of the components of the network in order to maintain not only the desired throughput but also to meet the standards and regulations designed to minimize the risk of high-pressure transmission lines.

Our research addressed the problem of determining the operational configuration of compressor stations that uses a minimum amount of energy (e.g. fuel, power) for given transportation requirements. This configuration will include, for example, the set of compressors that should be On/Off, and if On, the corresponding level of operation. The solution procedure should lead to the result in a short period of time to enable optimization of operations as often as necessary to keep the process as close as possible to optimal conditions.

Optimization of natural gas pipeline operations can be achieved by optimizing objectives such as fuel consumption, throughput and *linepack*[1], see [2], [3]. The objective of our research work was also to reduce the computational cost and therefore the time needed to find the vector of operational settings ($\vec{x}$) that optimizes the operation of a pipeline transmission network, i.e. achieve a minimal total cost of transportation of natural gas while satisfying safety regulations and meeting customer demands.

More formally, the main variables that affect the fuel consumption are the mass flow rate ($\dot{m}$), suction pressure (P_s) and discharge pressure (P_d) at each compressor station (CS). The cost of operation of CS_i can then be expressed as $\Psi_i(\dot{m}, P_s, P_d)$. The objective function $\vec{f}(\vec{x})$ for the fuel minimization problem can then be calculated as:

$$Minimize \quad \vec{f}(\vec{x}) = \sum_{i=1}^{n_{CS}} \Psi_i(\dot{m}, P_s, P_d) \tag{1}$$

where n_{CS} is the total number of CSs in the pipeline network and $\{\dot{m}, P_s, P_d\} \in \vec{x}$. Some of the constraints imposed to the solution of $\vec{f}(\vec{x})$ are governed by the physical characteristics of each compressor unit such as *surge*[2] and *stonewall*[3] limits, minimum and maximum speed, and minimum and maximum power.

Other sets of equality and inequality constraints are governed by pipeline characteristics such as operating pressure limits, mass balance equations and flow equations. Although a detailed description of these equations is out of the scope of this paper, a good reference for pipeline hydraulics can be found in [9].

In addition to the compressor settings described above, other hydraulic components that belong to $\vec{x}$ and affect the cost function $\vec{f}(\vec{x})$ are the settings of all control valves (CV) and block valves (BV) in the transportation system, as well as the availability of gas supply (receipts) and market demand (deliveries). It is worth noting that some of the components of the solution vector $\vec{x}$ are discrete decision variables such as the status of the CSs (On/Off) and the status of BVs (Open/Closed), hence adding complexity to the solution surface and as consequence difficulty to the problem.

Solving the optimization of the pipeline operations problem is definitely not straightforward due to the non-convexity of the feasible domain of compressor units, non-linearity and non-convexity of the fuel cost function, and non-convexity of

[1]Volume of gas contained in a pipeline system at any point in time.

[2]Limit set to avoid unstable conditions (pulsating flow) in centrifugal compressors operating under low flow conditions.

[3]High flow condition in which the velocity of the fluid can approach sonic speed.

the set defined by the pipe flow equations [14]. The dimension ($|\vec{x}| = n_x$) of the optimization problem directly depends on the size (n_{CS}) and configuration ($Conf$)[4] of the pipeline network under investigation and the number of parameters considered sufficient to define its operation. A typical network might consist of thousands of pipes, dozens of CSs with several compressor units inside, meter stations, cooling systems and a large number of different devices such as CVs and BVs.

The big challenge of this optimization problem is to identify the set of pipeline operational settings —e.g. pressure at control nodes, mass flow rates, compressors settings, status of block valves, etc.— that optimize objectives such as the fuel consumption in real time. The solution space of $\vec{f}(\vec{x})$ may be composed by many, from hundreds to millions, of combinations of operational settings that satisfy equality and inequality constraints but the goal of our work is to identify the combination of operational parameters in $\vec{x}$ (or sets of combinations) that optimizes $\vec{f}(\vec{x})$ in a timely manner. Solving this problem has proven to be computationally expensive. See [2], [3] for previous work in this area and the quality associated with the solution provided by each approach.

3. THE MULTI-AGENT SOLUTION APPROACH: HOMOGENEOUS TECHS

Multi-agent systems are a very good way to model and implement distributed search concepts. The following terminology is taken from [5]. A *multi-agent search system* $\mathcal{MASS}$ consists of a start agent $\mathcal{A}g_S$, an end agent $\mathcal{A}g_E$, a set of search agents $\mathcal{A}g_1$,...,$\mathcal{A}g_n$ and a communication structure $\mathcal{K}om$. We also have a set PU of processing units that are used by the agents to perform their work. A $\mathcal{MASS}$ is aimed at solving a search problem SP, more precisely its start agent $\mathcal{A}g_S$ takes an instance $Inst$ of SP and creates instances $Inst_1$,...,$Inst_n$ of search instances for the search agents. The search agents take their instances, work on them and communicate with the other agents using $\mathcal{K}om$ while doing so. If all search agents have finished their work (and all messages between agents have been received), then the end agent $\mathcal{A}g_E$ uses the results of all search agents to create the solution to $Inst$.

Obviously, the search agents are the most important components of $\mathcal{MASS}$. Each *search agent* $\mathcal{A}g_i$ is characterized by the triple ($\mathcal{P}r_i$,$\mathcal{K}om$,mes_i), where $\mathcal{P}r_i$ is a search process, mes_i the communication function of $\mathcal{A}g_i$ and $\mathcal{K}om$ the already mentioned communication structure. The search process $\mathcal{P}r_i$ is characterized by the triple ($\mathcal{A}_i$, $\mathcal{E}nv_i$, $\mathcal{K}_i$), where $\mathcal{A}_i$ is the search model used by the process, $\mathcal{K}_i$ is the search control that the process uses for this search model and $\mathcal{E}nv_i$ is the environment within $\mathcal{MASS}$ that the process is able to perceive. Since we are doing a search, this environment is a subset of the data areas in $\mathcal{K}om$. The control of a search process takes the current state s of the search by the process and the current values of the data areas in the environment e and then selects one state among all the possible successor states of s (as defined by the search model) as the next state of the process. Since e is considered by the search control, the search of a search agent can be influenced (outside of itself) by changing this environment, which is done by other search agents using their communication functions. A communication function mes_i takes the current values of the data areas in $\mathcal{K}om$ and the current search state of $\mathcal{A}g_i$ and creates new values for the data areas in $\mathcal{K}om$ (usually it creates only new values for some of the data areas in $\mathcal{K}om$).

How $\mathcal{K}om$ is structured and what the mes_is are allowed to do with $\mathcal{K}om$ is an important part of defining a particular multi-agent search system. By using a communication structure, we can abstract from the physical communication channels available to us (both blackboard-like approaches and message-passing approaches can be modeled), but it should be noted that implementing a particular $\mathcal{K}om$ on different hardware structures can lead to considerable differences in performance. This can also be true for the method in which the agents are mapped to PU. If performance is an issue (as in our case), then each active agent should have its own processing unit and only as many agents should be active as we have elements in PU.

The particular distributed search approach that we have chosen for our work is called TECHS (**TE**ams for **C**ooperative **H**eterogeneous **S**earch, see [7], although for this work we use only homogeneous search agents) and it is an approach that follows the "improving on the competition approach" paradigm for distributed search (see [5]). A multi-agent search system based on "improving on the competition approach" provides every search agent with the complete instance of the search problem that is to be solved. The pure competition approach then simply lets all agents search until one of them finds a solution. Improvements allow the agents to communicate while they are working on finding a solution.

In TECHS, the work of the search agents is done in so-called rounds, where each round consists of a phase in which each search agent tries to solve the given search instance using its search process. This is followed by a phase in which each agent evaluates its results in order to decide whether the new information it has found might be of interest to the other agents. Then the selected information is sent to the other agents (or selected other agents) who then evaluate all the received information again with regard to how useful this information might be for them and then integrate the information that passes this filter into their search for the next phase. Figure 1 shows one round in this general cycle for 3 agents.

More precisely, the start agent $\mathcal{A}g_S$ passes the given search instance $Inst$ on to all search agents who create start states for their search and then perform their searches for a given time (or a certain number of transitions to new states). The communication structure $\mathcal{K}om$ consists of a data area for each agent and each type of information that is communicated between agents. In [7], four different types of information were suggested: positive partial solutions, negative partial solutions, positive control information and negative control information. Note that partial solutions include full solutions and in our application we will only exchange full solutions. Solutions are directly incorporated into the search states of an agent, while control information affects the search control of an agent. In our application, the search processes of the search agents allow only positive information to be used (see the next section). But there are search processes that can make use of negative information, as documented in [7].

The selection of the information to be shared with other

[4]E.g. gun-barrel, serial, parallel, looped, etc.

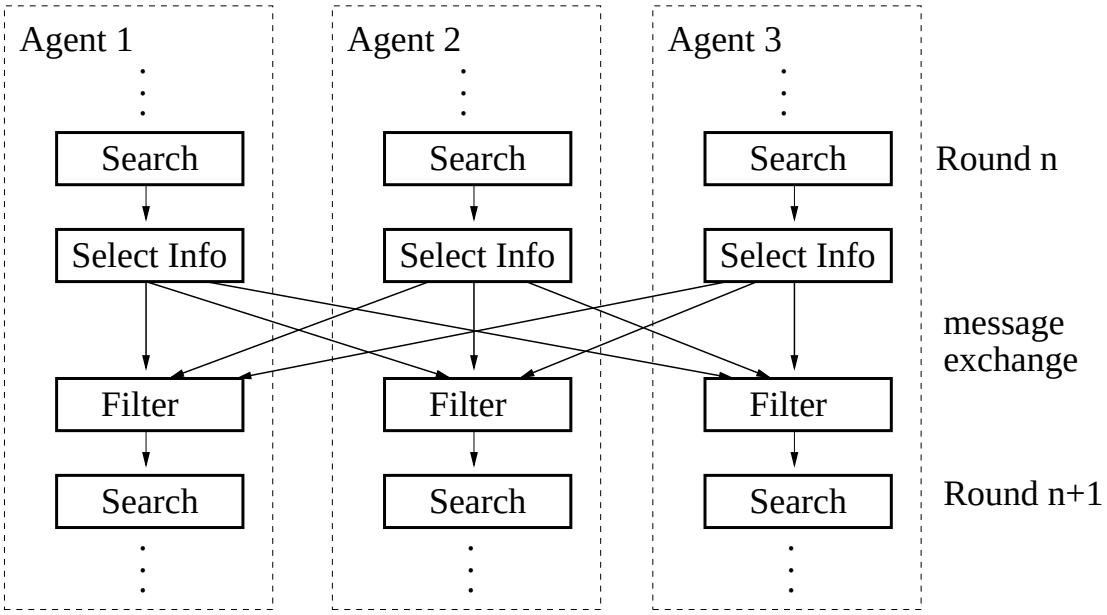

Figure 1: A round in TECHS for 3 search agents

agents is done by the *mes*-function of an agent, which can change all data areas of all other agents. Since each agent has its own set of data areas, it is possible to select different information (within a type) for different agents. The *mes*-functions for TECHS are usually implemented by sending messages to the other agents. The integration of the new information in the data areas for an agent $\mathcal{A}g_i$ (that form the agent's environment $\mathcal{E}nv_i$) is done according to the type of the information. As already stated, the solutions are appropriately added into the search state (if they pass the filter criterion), while the control information from then on influences the decisions of the search control of the agent.

The whole search ends if either the (optimal) solution is found (if it is possible for the search agents to detect this) or a certain time limit is reached. The end agent collects the results from all search agents and then either presents the found solution or the best result found in the given time. The TECHS approach is a general concept for organizing a distributed search system. It needs to be instantiated for a particular application and this instantiation centers around the search agents.

4. THE SEARCH AGENTS

Optimizing pipeline operations requires working with both, continuous and discrete variables. As a result, search methods that evaluate more and more instantiated partial solutions, like Branch-and-Bound or A*, are not easily applicable to this kind of problem. The fact that the quality of a solution requires a simulation to be determined rules such approaches completely out, since a partial solution usually just leads to an error from the pipeline simulator.

Therefore the search methods used for our problem are instantiations of what we call set-based search: a search state consists of one or several (a *set* of) solutions and there are operators that use all or some of the solutions in the current state to create new solutions. Examples for set-based search are hillclimbing, simulated annealing, tabu search (all these use only one solution in a state), genetic algorithms, evolutionary strategies or particle swarm systems (PSS). For our search agents, that form the IOPO system (**I**ntelligent **O**ptimization of **P**ipeline **O**perations), we have chosen to use PSS (see [12]).

As in the case of most of the set-based search methods, the start state of any of our search agents consists of a number of randomly created solutions (and each agent does this random creation on its own, resulting in different start states between the agents). In our experiments (see Section 5), we used 10 solutions (or particles, as they are called in PSS). A solution for an instance of the pipeline operations problem consists of the parts of the vector $\vec{x}$ (see Section 2) that describe for each compressor station CS_i in the instance $Inst$ its status $stat_i$ (On/Off) and its control pressure (either as P_{s_i} or P_{d_i}), for each block valve BV_i its status $Bstat_i$ (Open/Closed) and for each control valve CV_i its control pressure (either as P_{s_i} or P_{d_i}, again). An instance $Inst$ of the problem itself consists of the network topology (also called network configuration $Conf$), that is needed for the simulator and that provides the CS_is, BV_is and CV_is, flow requirements at specific points of interest in the pipeline network and a set of boundary conditions for the network that describe the receipts and deliveries (i.e. location and volume of gas that goes into or out of the network). In a PSS, we also create for each particle p_i a so-called velocity, which in our case is a velocity vector $\vec{v}_i$. Again, this vector is initially created at random.

The search control of an agent stores the best solution ($\vec{x}_B$) found so far. A search agent creates its next search state in the following manner (if its environment has not changed): for each particle p_i, which represents the solution $\vec{x}_i$ and the velocity $\vec{v}_i$, we also remember its best ancestor solution $\vec{x}_{BP_i}$. Then we create for each p_i its successor particle (which we will call also p_i) as follows. The new solution $\vec{x}_i^{new}$ is computed by

$$\vec{x}_i^{new} = \vec{x}_i + \vec{v}_i. \tag{2}$$

If this solution is a valid solution and better than $\vec{x}_{BP_i}$ (using the network simulation to evaluate the gas consumption of this solution to determine its validity and quality), then $\vec{x}_{BP_i}^{new} = \vec{x}_i^{new}$, else $\vec{x}_{BP_i}^{new} = \vec{x}_{BP_i}$. Finally,

$$\vec{v}_i^{new} = W\vec{v}_i + C_1 r_1(\vec{x}_{BP_i} - \vec{x}_i) + C_2 r_2(\vec{x}_B - \vec{x}_i), \qquad (3)$$

where W is a weight parameter controlling the influence of the previous velocity, C_1 is the so-called *cognitive learning factor*, C_2 the so-called *social learning factor* and r_1,$r_2 \in [0, 1]$ are random values chosen by the search control. So, we have the ancestors of a particle influence it two ways (via the old velocity and the best ancestor, so far) and the rest of the particles influence it in form of the best particle overall. Note that the elements of $\vec{x}$ that represent the discrete variables will only be transformed into discrete values (by rounding) for the simulator, within a particle we work with real values.

With several agents, their cooperation leads to some modification of how the next state of a search agent is computed. Let us first look at what the agents communicate to each other. As stated in the last section, for PSS we have only found ways to make use of positive information. This means that $\mathcal{K}om$ contains for each agent $\mathcal{A}g_i$ two data areas, let us call them $poscon_i$ and $posst_i$. In $poscon_i$, the other search agents' mes_j-functions put their best solution found so far, i.e. $\vec{x}_{B_j}$, when they select the information to be communicated after a round of search. In $posst_i$, the other search agents' mes_j-functions put the best k_1 solutions they found during the last round (in our experiments, k_1 was 2). While at first glance the information in $poscon_i$ seems to be redundant, this is not the case for two reasons. Both reasons are related to the usage $\mathcal{A}g_i$ makes of this information in its environment.

Firstly, the information in $poscon_i$ is filtered by choosing from it the best solution overall and then this solution replaces the agent's $\vec{x}_{B_i}$ (if it is better than the agent's previous best solution). This means that we have changed the search control of the agent (that now selects a different velocity adjustment). The information in $posst_i$ is also filtered by selecting the best k_2 solutions in $posst_i$ (in our experiments, k_2 was 3). The selected k_2 solutions are then used to replace the k_2 worst current solutions in $\mathcal{A}g_i$'s state, essentially moving the "position" of these k_2 particles to different (and hopefully better) places. The velocities of the particles are not changed with this substitution and the best ancestor of each particle is only updated if the new solution is better. By not touching the velocities, different agents might use the same solutions from their $posst_i$s, but will produce different successors for the effected particles, if the velocities were different. So, the use agents make of their $poscon_i$ and $posst_i$ data areas is different.

But the second reason makes the need for two areas even more obvious. New values for the $posst_i$s are not produced after every round, only every few rounds (where "few" in our experiments meant every 3 rounds). Updating the search states of the agents after every round would quickly achieve that the agents search the same area of the search space, thus resulting quickly in a lot of redundant search steps. While the usage of the information in the $poscon_i$s only establishes a common "direction" for the particles, using the $posst_i$'s after every round will move k_2 particles to or near this best solution, without exploring solutions situated between the current positions and this best area. It should be noted that making use of $posst_i$ has also positive effects, namely "rescuing" search agents that explore bad local optima and have their particles converged to this optimum by moving some particles elsewhere. Balancing the positive and negative effects of the use of $posst_i$ requires some calibration of the system, which is how we determined the number of rounds between using $posst_i$ in our experiments.

There is one additional aspect of our search agents that needs to be mentioned, namely how they make use of the hydraulic simulator that provides them with the evaluation of the quality of a solution, respectively identifies solutions that are not valid. The simulator used by our system is proprietary to TransCanada and therefore had to be treated like a blackbox. More precisely, each search agent starts the simulator in an own process each time it needs to evaluate a solution. The solution is written in a file that is given to the simulator as a command-line parameter and the simulator writes its result in another file that is consulted by the search agent after the simulator process terminated. This is far from optimal and causes some problems regarding effective use of the given hardware platform, but we were still able to produce good results.

5. EXPERIMENTAL EVALUATION

In this section, we summarize the results of a series of experiments demonstrating the performance of the IOPO system when applied to the problem of optimizing TransCanada's pipeline operations. In addition, we present the effect that the use of a multi-agent search system has, especially with regard to multi-core multi-processor technology as an underlying hardware platform.

Research on optimization of pipeline operations is a very competitive area, as significant improvements in operations can generate commercial advantage over competitors. Therefore we cannot reveal detailed information so as not to compromise any advantages that TransCanada currently has. This means that we are only allowed to report on a limited number of pipeline sub-networks, we cannot reveal the real cost of solutions, and the level of detail about the pipeline system studied in these experiments has been restricted to a very high level description. Due to the proprietary nature of the information, further detail cannot be disclosed in this paper.

Existing software tools are unable to handle block valves which have only two states, fully opened or fully closed. In addition to this, the limited number of available decision variables restrict the area of the network that can be simulated. As consequence at TransCanada, the established practice involves the use of in-house developed simulation software to analyze pipeline operations and determine the operating configuration that will optimize fuel consumption on the pipeline system. After an initial solution is obtained, this solution usually is not directly applicable, so the user needs to make adjustments to the input and repeat the process several times with adjusted input before an acceptable solution is found. A major shortcoming of the established method is the significant amount of human interaction and intervention and therefore the significant time and effort required to successively iterate to a solution.

Due to these problems, TransCanada has been actively researching methods and evaluating commercial products to improve upon the current practice. TransCanada supported the research presented in [2] and [3], which developed the MOGA method that used Genetic Algorithms to

Table 1: Comparison of the three systems with regard to run time (wall clock time)

Instance	MOGA	Commercial Method	IOPO
SN1/HF	0:25:00	0:05:31	0:04:19
SN1/MF	0:26:15	0:05:38	0:02:32
SN1/LF	0:23:00	0:05:27	0:01:59
SN2/HF	107:46:00	0:43:09	0:29:02
SN2/MF	107:49:01	0:49:43	0:27:06
SN2/LF	99:14:00	0:49:09	0:28:19

Table 2: Comparison of the three systems with regard to solution quality (Commercial Method = 100%)

Instance	MOGA	Commercial Method	IOPO
SN1/HF	101	100	99
SN1/MF	104	100	104
SN1/LF	102	100	102
SN2/HF	90	100	86
SN2/MF	96	100	88
SN2/LF	92	100	86

solve the problem fully and automatically. However, this research was not able to provide the run-time performance that TransCanada needs. This research used hydraulic simulation software developed in-house at TransCanada. Our work also uses the same hydraulic simulator.

In addition to the MOGA research activities, TransCanada has been evaluating the suitability of commercial optimization software for the purpose of optimizing fuel usage on the pipeline system. For the purpose of this paper the commercial software is referred to as the *commercial method*. Similar to the in-house developed method, shortcomings have been encountered with the commercial method of fuel optimization: it too requires a significant amount of human intervention to successively iterate to a solution, the user needs to find a hydraulically valid and feasible simulation result as a starting point for the optimization software and some pipeline components, such as block valves, cannot be modeled as decision variables in the optimization method. We will use this commercial method as comparison point in our experiments to reflect the state-of-the-art in the field and to add to the protection of TransCanada's proprietary information. Neither the commercial nor the MOGA optimization methods have been developed to take advantage of the multi-core hardware.

The experiments we report in the following have been run on a 2 Dual Core Xeon 5160 3.00 GHz workstation with 2 GB of RAM running Windows XP as the operating system. Each search agent in IOPO has its own process, with one of these processes also realizing the start and end agent. The communication structure $\mathcal{K}om$ was done by having the mes_i send messages between the processes. IOPO is implemented in C.

5.1 Comparison of IOPO with other systems

Our first series of experiments is documented by Tables 1 and 2. We looked at two different pipeline subnetworks within TransCanada's pipeline network. These subnetworks, SN1 and SN2, represent the smallest subnetwork (SN1), with 6 compressor stations and one control valve resulting in 13 decision variables, and one of the largest subnetworks (SN2), with 20 compressor stations, 8 control valves and 3 block valves, resulting in 53 decision variables. Both subnetworks represent actual divisions of the network that are optimized separately in TransCanada's current practices (naturally, the subnets connect to other subnets, but the boundary conditions between the subnets are negotiated by human operators). For each subnetwork, we have 3 instances of flow requirements, namely a high flow (HF), a low flow (LF) and a medium flow (MF). For the wall clock time for the commercial method we added to the system run time the estimated worst case interaction time for an experienced user, which was for SN1 5 minutes and for SN2 40 minutes. For IOPO, we report in both tables the average over 10 runs (due to the heavy use of random factors in the search). While MOGA also uses random factors, we performed only one run for each entry, due to the very long runtimes for SN2.

Table 3: Runtime comparison for different numbers of agents for SN2 instances (in minutes, on 2 Dual Core machine)

Instance	1 agent	2 agents	3 agents	4 agents
SN2/HF	54	33	31	30
SN2/MF	50	29	28	27
SN2/LF	56	33	31	29

As Table 1 shows, IOPO and the commercial method need approximately the same time to produce their results, IOPO fully automatically, not requiring a user providing a valid solution and using 4 search agents (one for each available core). MOGA's run time for large networks is far too long to meet TransCanada's requirements to support pipeline operations on a day-to-day basis.

Results to measure the solution quality in terms of fuel consumption were normalized using the commercial method as reference (100%). Note that this means that smaller values represent better solutions. If we look at the solution quality in Table 2, then the three systems are quite comparable for the small network SN1. But IOPO is substantially better for the large network (that naturally requires much more gas in its operation than the small network). For the SN2/MF instance the IOPO fuel consumption is 12 percent lower than the commercial method which would amount to an operating cost reduction of $ 28.7 Million per year for the combined cost of fuel gas and the environmental impact of CO_2 emissions, at current market prices.

5.2 Evaluation of the multi-agent/hardware platform aspect

Our next series of experiments aim at evaluating the multi-agent aspect of our IOPO system, more precisely the influence of the number of agents used on the results. We concentrate our analysis only on the search instances for SN2, since the runtimes for SN1 are too short to see any significant changes. As stated in our introduction, a big draw towards using multi-agent search approaches for hard optimization problems in industry is the seemingly obvious fit for multi-processor multi-core workstations in order to use the number of available processors to speed-up the search. In order to analyze this aspect of our system, we run experimental series for 1,2,3 and 4 agents in IOPO where for

Table 4: Runtime comparison for different numbers of agents for SN2 instances (in minutes, on network of 4 Pentium IIIs)

Instance	1 agent	2 agents	3 agents	4 agents
SN2/HF	240	125	92	69
SN2/MF	220	116	82	63
SN2/LF	225	132	83	65

each number of agents the total number of solutions created (and therefore the number of calls of the simulator) were the same, namely 4000 (which was also the number used for the experiments in the previous subsection). This means that with one agent this agent does 400 updates of the 10 particles, with 2 agents we have each agent do 200 updates of their 10 particles and so on. For 2,3 and 4 agents we fit their searches into 10 rounds, so that for two agents the communication between the agents takes place every 20 updates up to communicating after every 10 updates for 4 agents. This way, the communication overhead is essentially the same for each number of agents and we can get a good picture regarding the utilization of the processors by IOPO. As in the previous subsection, we report the average runtime over 10 runs.

Table 3 shows the results of the experimental series described above. As can be seen, the results are rather disappointing. While using 2 agents gives us quite some speed-up, adding the 3rd and the 4th agent does not really accomplish a lot. The question from the multi-agent perspective is now, if this disappointing outcome is due to the multi-agent search approach, i.e. TECHS, or due to the combination of this approach with a multi-processor multi-core hardware platform. While we did not use all of the features of TECHS as described in [7] (by not using heterogeneous search agents we definitely reduced the possibility for synergetic effects), and it is a well-known fact that for each cooperative search scheme and each problem instance there is a maximum number of processors beyond which no gains in speed can be achieved, we nevertheless were not convinced that this number is 2 for all the search instances we were looking at. Therefore we suspected the hardware platform to be the problem and in order to prove this we switched to a network of 4 old Pentium III machines (with 256 MB of RAM each) that we had available.

Table 4 presents the results with these Pentium III machines. As can be seen, adding a 3rd and 4th agent (i.e. processor) now has still quite an impact on the runtimes, ruling out the TECHS approach as sole problem. We can not expect linear improvements when adding more agents/processors in our setting, simply because with a smaller number of particle updates the mes_i-function of an agent does have to choose from less results, thus providing less useful information to other agents, respectively really useful information later during a run. Therefore the results of Table 4 reflect well the behavior we expect from a homogeneous version of TECHS.

But why does changing from the multi-processor multi-core platform to a multi-computer platform produce a so different behavior? While we were able to assign the search agents to different cores, we do not know what the operating system does with the simulator processes. But even more important, all cores do share the same periphery, i.e. memory that is not on the processor chip and access to the file system. Given the fact that the pipeline simulations produce substantial file traffic and also require significant memory, this might produce system level bottlenecks that result in the observed disappointing behavior. This will require additional research and at the moment we have to accept the fact that we might have to include systems level programming into the search agents to overcome this. And therefore we have to be careful what we promise industry when suggesting the use of multi-agent search for hard optimization problems.

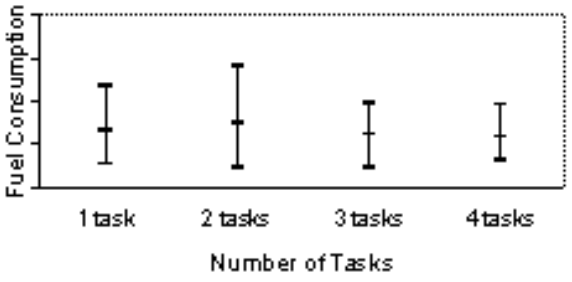

Figure 2: Variance in solution quality over 10 runs for different numbers of agents (HF instance)

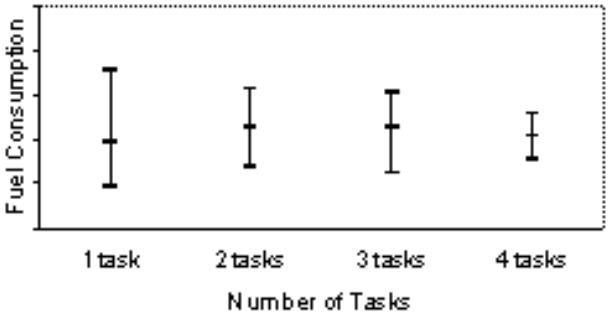

Figure 3: Variance in solution quality over 10 runs for different numbers of agents (MF instance)

While the utilization of the available processors is not a strong point of IOPO (at least for a multi-processor multi-core hardware problem), speed-up is not the only thing that is of interest for our industry partner. Most set-based search approaches make considerable use of random factors, so that two runs of a system using such approaches usually lead to two different solutions being found as the best solution of a run. How much runs vary with this regard is of quite some importance for a user. Therefore we compared the results from Table 3 with regard to how much the runs vary in their solution quality.

Figures 2, 3 and 4 give a graphical representation of the variance in solution quality in our experiments[5]. As the figures show, using more agents has a positive effect here for all number of agents, since the variance is reduced with each

[5]Again, we have to conceal the real values of the solutions to not endanger TransCanada's competitive edge; the x-axis does not meet the y-axis at y=0 in these figures

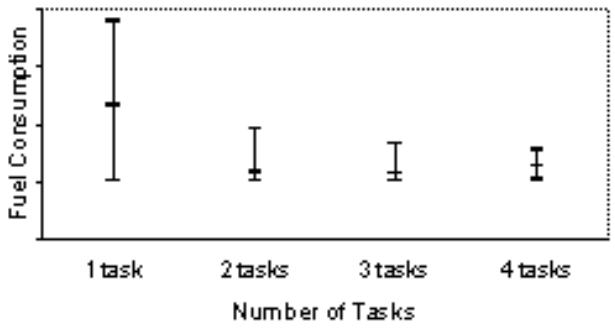

Figure 4: Variance in solution quality over 10 runs for different numbers of agents (LF instance)

agent added to the system. This means that the quality of the solutions produced by IOPO is more predictable with more agents used, which from an economical and planning point of view is a very important observation.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented an application of a homogeneous version of the TECHS concept for multi-agent cooperative search to the problem of optimizing pipeline operations. Our resulting system, IOPO, was able to solve several instances of a complex pipeline subnetwork of TransCanada with at least 10 percent less consumption of gas for operations, fully automatically, in a time comparable to the time needed by the commercially available system that requires human interaction. The use of a multi-agent approach also showed to be advantageous with regard to reducing the solution quality variance between several runs of the system, which is an important criterion for industrial users of search systems.

While the current trend in hardware towards multi-processor multi-core workstations could serve as an attractor for industry to multi-agent concepts, our experiments show that such an argument can have problems for some applications, since there is significant potential for bottlenecks on the systems level. A cluster of workstations still seems to present a more suitable platform for applications with a profile like our IOPO system.

The research reported in this paper is just a first step in exploiting the potential that multi-agent cooperative search has for solving hard optimization problems in industry. The previous applications of the TECHS concept (see [7] and [6]) reported synergetic effects for the case of heterogeneous agents. While there are no tree-based or graph-based search methods that can be applied to our application problem, other set-based search approaches could provide enough heterogeneity to boost IOPO's performance. This research will be continued to use IOPO instances for different connected pipeline subnetworks of the TransCanada system. The results would reinforce the viability and robustness of IOPO, and the potential for deployment of the system.

7. ACKNOWLEDGMENTS

The authors wish to thank TransCanada Pipelines Limited for the valuable support and resources provided throughout the course of this research work.

8. REFERENCES

[1] T. Bäck: Parallel Optimization of Evolutionary Algorithms, Proc. Parallel Problem Solving from Nature III, LNCS, 1994, pp. 418–427.

[2] K.K. Botros, D. Sennhauser, K. Jungowski, G. Poissant, H. Golshan, and J. Stoffregen: Multi-Objective Optimization of Large Pipeline Networks Using Genetic Algorithms, Proc. Int. Pipeline Conference, Calgary, 2004 (on CD).

[3] K.K. Botros, D. Sennhauser, J. Stoffregen, K.J. Jungowski, and H. Golshan: Large Pipeline Network Optimization - Summary and Conclusions of TransCanada Research Effort, Proc. Int. Pipeline Conference, Calgary, 2006 (on CD).

[4] J. Denzinger: Knowledge-Based Distributed Search Using Teamwork, Proc. 1^{st} ICMAS, San Francisco, 1995, pp. 81–88.

[5] J. Denzinger: Conflict Handling in Collaborative Search, in Tessier, Chaudron, Müller (eds.): Conflicting Agents: Conflict management in multi-agent systems, Kluwer, 2000, pp. 251–278.

[6] J. Denzinger and D. Fuchs: Cooperation of Heterogeneous Provers, Proc. IJCAI-99, Stockholm, Morgan Kaufmann, 1999, pp. 10–15.

[7] J. Denzinger and T. Offermann: On Cooperation between Evolutionary Algorithms and other Search Paradigms, Proc. CEC-99, Washington, IEEE-Press, 1999, pp. 2317–2324.

[8] C.G. Diderich and M. Gengler: Solving Traveling Salesman Problems Using a Parallel Synchronized Branch and Bound Algorithm, Proc. High-Performance Computing and Networking, Brüssel, 1996, pp. 633–638.

[9] M. Mohitpour, H. Golshan and A. Murray: Pipeline Design & Construction: A Practical Approach, ASME, 2003.

[10] M.J. Quinn: Analysis and Implementation of Branch-and-Bound Algorithms on Hypercube Multicomputers, IEEE Trans. Computation, Vol 39(3), 1990, pp. 384–387.

[11] J. Rachlin, R. Goodwin, S. Murthy, R. Akkiraju, F. Wu, S. Kumaran and R. Das: A-Teams: An Agent Architecture for Optimization and Decision-Support, Proc. ATAL-98, Paris, 1998, pp. 261–276.

[12] M. Reyes-Sierra and C.A.C. Coello: Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art, Journal of Computational Intelligence Research, Vol 2, 2006, pp. 287–308.

[13] S.N. Talukdar, P.S. de Souza and S. Murthy: Organizations for Computer-based Agents, Journal of Engineering Intelligent Systems, Vol 1, 1993, pp. 75–87.

[14] S. Wu: Steady-State Simulation and Fuel Cost Minimization of Gas Pipeline Networks, PhD Thesis, Department of Mathematics, University of Houston, 1998.

[15] M. Yokoo: Distributed Constraint Satisfaction, Springer, 2001.