D-WMS: Distributed Workforce Management using CLP

FOTIOS KOKKORAS
Department of Informatics
Aristrotle University of Thessaloniki
Thessaloniki, 54006, GREECE

Tel.: +30 31 998433
kokkoras@csd.auth.gr

STEVE GREGORY
Department of Computer Science
University of Bristol

Merchant Venturers Building,
Woodland Road
Bristol, BS8 1UB, UK
Tel.: + 44 117 9545142
steve@cs.bris.ac.uk

Abstract

We present a distributed CLP-based approach for solving a real workforce management problem (BT's DT-250-118). The problem consists of a set of jobs that we want to assign to engineers in order to serve as many of them as possible at a minimum cost. We first divide the problem into sub-problems and then assign each of them to a solving agent. Each agent works independently to solve its own sub-problem and then co-operates with its peers to optimise further the intermediate results. In the sub-problem solving stage, our agents use a CLP based approach which has been used in the past in a centralised, global way. Our method allows naturally distributed scheduling and resource allocation problems to be solved in a short time with minimal disruption to the quality of solutions when compared against global approaches.

Keywords: constraint logic programming, distributed problem solving

1. Introduction

Discrete optimisation problems, of the kind that arise in many areas of operational research (OR), can be modelled and solved using integer linear programming (ILP - linear constraints and integer variables). A fundamental difficulty with linear programming in real world optimisation problems is to find a compact representation for them. This increases the complexity of any given solution, particularly when the number of constraints to deal with grows or when real (or quazi-real) time response of the system is required. Another approach is to formulate those problems as constraint satisfaction problems (CSPs). A CSP consists of a set of variables, each with a finite set of possible values, and a set of constraints which the values assigned to the variables must satisfy. The optimisation part is handled by an additional variable representing the objective which we want to minimise or maximise.

Unlike ILP, the CSP approach provides greater expressiveness because constraints are not restricted to linear inequalities as in linear programming. As a result, a CSP representation requires far fewer constraints and variables than an ILP.

A typical example of a problem that fits the above model is the British Telecom (BT) Workforce Scheduling Problem (WSP). In similar studies on labour scheduling, an heterogeneous workforce model is assumed. That is, the workforce is a mixture of full-time and part-time employees with different wage rates. The aim is to reduce the wages of employees. In BT's case, all employees are full-time and the aim is to serve as many jobs as possible within a fixed time minimising the travel time between job locations. This is a heavily constrained optimisation problem and like many real world problems of that kind is NP-hard. Even from the CSP perspective, their huge state space prohibits the use of complete methods such as *branch and bound*. Recently, among other techniques [1,2,15,19], Constraint Logic Programming (CLP) coupled with heuristics has been successfully used to tackle with that problem in a very short time, giving a satisfactory solution without the need to perform a full search [20].

All the approaches so far to solve BT's WSP are global, that is, the whole problem is handled by a single solving process, although parallel processing methods have been successfully applied. The drawback of any global approach is that it requires all the problem related information to be collected in a specific place. This requires certain communication cost, which could be prohibitively high. In addition, in some cases, gathering all information to one location is not desirable or impossible for security reasons. In such cases a distributed approach is more appropriate.

The advances in hardware and networking technologies during the current decade have led to a very rapid spread of distributed computing environments. Distributed Artificial Intelligence (DAI) is a subfield of AI that is concerned with the interaction of *software agents*. A software agent is a self contained problem solving entity which is able to interact with other agents in order to complete its own problem solving and to help others with their activities. In [22], a distributed constraint satisfaction problem (DCSP) is formalised as a CSP in which variables and constraints are distributed among software agents. Although algorithms for solving DCSPs seem to be similar to parallel/distributed processing methods for solving CSPs [25], there is an additional research motivation, apart from the efficiency. In DCSPs there exists a situation where knowledge about the problem (i.e. variables and constraints) is logically or geographically distributed. In such cases multiple agents have to co-operate to solve the problem without centralising all information.

In this paper we show that the BT's problem is highly distributed by nature and we propose a method to solve it in a distributed way. We introduce two clustering methods for partitioning the problem into sub-problems and then we assign each of them to an agent that utilises the CLP method described in [20]. All agents work independently on their own sub-problem to find a first solution, and then communicate with their peers to further optimise the global solution. Our aim is to find out how efficient, in terms of execution time and solution quality, is such an approach.

The rest of this paper is organised as follows. Section 2 gives some background pertaining to Co-operative Problem Solving (CPS) and Distributed Constraint Satisfaction Problems (DCSPs). Section 3 states BT's workforce scheduling problem and reviews the approach used in [20]. Our distributed approach is covered in Section 4. Related work is presented in section 5. Section 6 concludes the paper and gives suggestions for future work.

2. CPS and Distributed CSPs

2.1 Co-operative Problem Solving

Finding solutions for large problems is sometimes most advantageously carried out as the joint responsibility of multiple agents. For example, a multiple agent approach can be essential when the areas of expertise relevant for solving the problem do not reside in any single agent but are found among multiple heterogeneous agents. Alternatively, a group of identical or similar agents might be employed for the purpose of finding a solution faster than it can be accomplished by a single problem-solver. Such a multi-agent system might be one in which the agents all work in parallel on the entire problem and share hints with each other, or it might be one in which the problem is subdivided among the agents. In the latter case, each agent works on solving its part of the entire problem, within some framework of interaction with the other agents, and eventually the subproblem solutions are recombined in some way to solve the overall problem.

When a problem is subdivided among multiple agents, the way in which the agents find, share and use partial results can greatly affect the overall efficiency of the problem solving effort, either positively or negatively. If all communication of subproblem solutions is postponed to the termination of problem solving, incompatibilities might force some or all of the agents to redo work already done. Had the agents communicated their partial results earlier, they might have used each others' partial solutions to direct their efforts, so that upon completion the final solutions were globally consistent.

On the other hand, early communication of subproblem solutions or partial solutions can steer other agents in a counterproductive direction. If a communicated solution that is ultimately inconsistent with any global solution is incorporated by another agent and used to guide its problem solving, that agent can spend a lot of time until it discovers that there is no solution in that direction. Alternatively, a consistent solution might be found, but it might be of lower overall quality than if the agent had worked more independently earlier.

2.2 Distributed CSPs

A DCSP is the union of a set of constraint networks, plus an additional set of inter-agent constraints. It consists of both an objective, centralised view, and subjective, agent-level views. We must represent explicitly what parts of the problem belong to which agents.

A DCSP can be represented as a distributed constraint network $CN=\langle A,V,D,C\rangle$, where:

- $A=\{a_1, a_2, \ldots a_m\}$ is a set of m variables,
- $V=\cup V_i$, $(i\in A)$, where Va_i is the set of variables belonging to agent a_i,
- $D=\cup D_i$, $(i\in A)$, where Da_i is the set of domains belonging to agent a_i
- $C=\cup C_i \cup \{c_i \mid c_i(v_{i1},..,v_{ij}) \subseteq D_{i1}\times\ldots\times D_{ij} \wedge \forall(v_{ik}\in V_l)\exists v_{im} \mid v_{im}\notin V_l\}$ $(i\in A)$

That is, distributed constraint network (DCN) is the union of the constraint networks of the set of A agents, plus additional constraints between agents. We further require that the sets of variables belonging to different agents be disjoint, that is, $\forall(i\neq j)\ (V_i \cap V_j = \emptyset)$. This is necessary in order to clarify what constraints are visible to what agents.

We define a *subproblem* to be a set of one or more variables that "belong together", either logically in the problem domain or because of the nature of the constraint relationships among them. All variables in a single subproblem must belong to the same agent, and the subproblems of a DCN partition the DCN's variables. An agent may be responsible for more than one subproblem. Problem decomposition, namely the allocation of variables to subproblems and subproblems to agents, is assumed to be predetermined and not subject to change through negotiation. This does not preclude dynamic creation of new variables or subproblems, as long it is clear who they belong to.

It is assumed that each agent a_i has full knowledge of those constraints $c_{j..k}$ where $\{v_i...v_k\} \subseteq V_{ai}$ and *directional* knowledge of those constraints $c_{j..k}$ for which some $v_j \in V_{ai}$ and some $v_l \notin V_{ai}$. Directional knowledge of constraints means that an agent can evaluate how assignments made to non-local variables with which it shares one or more constraints affect the domains of its own variables, but it cannot evaluate how assignments it makes to its own local variables will affect the domains of non-local variables.

Research efforts in centralised constraint satisfaction [5,6,7,9,13,14,17] have been extended to a distributed context. Some of them are outlined next.

2.2.1 Constraint Partition and Co-ordinated Reaction

In this methodology problem constraints are partitioned based on their type, clustered based on their connectivity, and distributed to agents specialised for the type [10,11]. Some heuristic co-ordination strategies are used to help promote rapid convergence to a solution:

- *Least disturbance*: change as few variables as possible;
- *Island of reliability*: these are constraint clusters whose variables have the least flexibility, and provide an anchor for problem solving;
- *Loop prevention*: use counting and history to prevent looping behaviour.

2.2.2 Dynamic Value and Variable Ordering Heuristics

Nishibe, Kuwabara and Ishida [16] introduce and evaluate one value and one variable ordering heuristic for distributed problem solving. The value ordering heuristic has to do with probability of conflict of an entire local solution, not just of one variable, and is computed as follows: each agent computes all of its possible local solutions and sends them to all other agents with whom it shares constraints (neighbours). Each agent then evaluates the conflict probability between each of its own local solutions and the received partial solutions from all neighbours, and finds the highest probability of conflict for each local solution. Then each agents orders its own solutions in reverse order of probability of conflict.

In the case of the variable ordering heuristic, "variable" means a single agent's local solution. However, many local variables are involved and variable ordering refers to which agent will produce an alternate solution when a conflict arises. Thus, this heuristic actually gives rise to an agent ordering through a probability value of conflict. When used in conjunction with distributed asynchronous backtracking [24], this measure is computed once, at the start of problem solving, and a static agent order is derived, as necessary to guarantee completeness of the algorithm. When used in conjunction with hill-climbing the measure is computed dynamically, at every conflict, based on what local solution is being considered at the time. In this case it shows how tight an agent's constraints are when it is assigning a particular local solution. In the hill-climbing method, in the case of a conflict between the local solution of two agents, the agent with the lower conflict value, which is more loosely constrained, will select an alternative solution. This method does no backtracking.

Yokoo [23], presents and evaluates one value and one variable ordering heuristics to increase asynchronous problem-solving efficiency. These are:

- *min-conflict*: when selecting a value for a variable, select a value that minimises the number of conflicts, and
- *change-oldest*: when selecting a variable to change from among conflicting variables, select the one that has not changed in the longest period.

How the success of these heuristics depends on the problem class on which they are used, was not evaluated, though. Furthermore, the communication cost is not taken into account. When a process backtracks, this must be propagated around the network in order for variable priorities to be updated. Communication cost is a very important issue in distributed problem solving as it affects the time taken to solve the problem.

2.2.3 Distributed Forward Checking and Conflict-Directed Backjumping

Luo, Henry and Buchanan in [12] have adapted several backtracking improvement techniques from centralised constraint satisfaction, namely forward checking, constraint learning and backjumping, for use in distributed problem solving. These algorithms include the necessary inter-agent communication, while employing methods to try to keep it to a minimum.

In comparisons of several of these distributed algorithms with parallel and sequential counterparts, the distributed ones tended to do worse, on average.

3. The BT Workforce Scheduling Problem

3.1 Problem Statement

The workforce scheduling problem is to determine how to assign jobs to employees so as to maximise the amount of work done and to minimise the amount of travel. It can be considered as a multi-TC-TSP (Time Constrained Travelling Salesman Problem). A TSP involves selection of the shortest route among a number of locations to be visited, subject to the constraints that the person must return to the starting point at the end of his/her journey, and can visit each location only once. A time constrained TSP involves additional time related constraints that need to be satisfied (e.g. the journey must be finished before a certain time, some jobs must be done in the afternoon, etc.). The data set we worked on is known as RD-250-118 [3] and is described next.

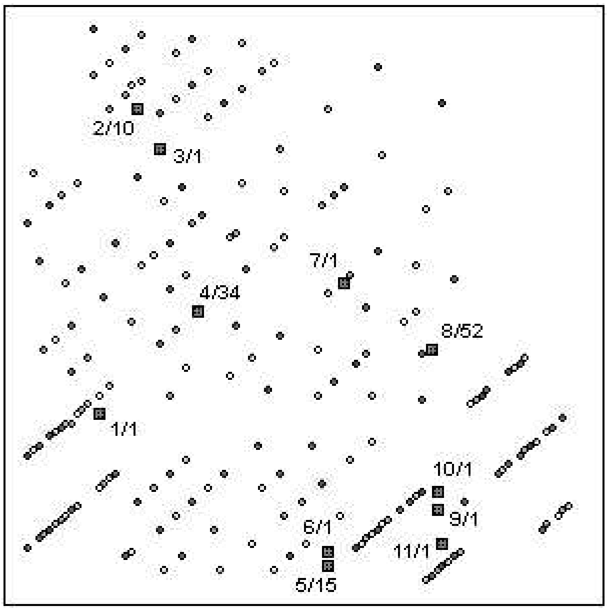

Figure 1: Location of jobs and bases for the RD-250-118 dataset. The numbers next to each base indicate the BaseID and the number of engineers that belong to that base, respectively.

Jobs: There are 250 jobs, each one defined by the 5-tuple:

`(JobID,JobX,JobY,Duration,Type)`

The pair of co-ordinates `(JobX, JobY)` indicate the location of the job. `Duration` is the time needed by an average skilled engineer (see later) to accomplish the job. Job types are: *first/last* (must be the very first/last in a tour, respectively), *morning/ afternoon* (must start before/after 12:00 respectively), and *any_time* (can be done at any time). The distribution of the jobs over the problem's area is displayed in Figure 1.

Bases: There are 11 bases represented by a triple: `(BaseID,BaseX,BaseY)`. The pair of co-ordinates `(BaseX,BaseY)` indicates the location of a base. The location of the bases and the number of engineers they own are also displayed in Figure 1

Technicians: There are 118 technicians. Each one is defined by the 6-tuple:

`(TechID, BaseID, StartTime, EndTime, Skill, Overtime)`

Thus, the engineer belongs to a base, starts and finishes every day at a certain time, has a skill factor and is allowed to work a certain duration overtime (like in earlier work on this dataset, we do not allow overtime). The base of a technician points to a pair of co-ordinates which is actually the co-ordinate pair of the base. The start time is either 8:00 or 8:30 while the end time is either 16:00 or 16:30 respectively. The ratio `Skill`/10 indicates the fraction of the standard duration that this technician needs to accomplish the job. For example, a 30-minute job is served in 24 minutes by a technician with skill factor 8 (30*8/10=24).

Skill Constraints: For each job, there is a list of engineers associated with it that indicates which of them are qualified to do the job. From now on, we will call this list the *job's technician list*.

Travel Time: The travel time between two locations `(X1, Y1)` and `(X2, Y2)` is given by the following function [1]:

$$\frac{|X1-X2|+|Y1-Y2|/2}{8} \text{ if } |X1-X2|>|Y1-Y2|\ , \quad \frac{|Y1-Y2|+|X1-X2|/2}{8} \text{ otherwise}$$

Cost Function: The quality of a solution is calculated by the following cost function:

$$Cost=\sum_{i=1}^{Techs} TravelTime_i+\sum_{j=1}^{Jobs}(Duration_j+Penalty)*Flag_j$$

where: *Techs* is the total number of technicians, $TravelTime_i$ is the total travel time of engineer *i*, *Jobs* is the total number of jobs, $Duration_j$ is the duration of job *j*, *Penalty* is a constant, currently set to 60 and $Flag_j$ is 0 if job *j* is allocated to an engineer; 1 otherwise. Strictly speaking, what we want to do is to minimise the cost function.

3.2 The WMS

The approaches that have been used so far to tackle the stated problem include simulated annealing [2], genetic algorithms [15], fast local search and guided local search [19] and finally, constraint logic programming [1,20]. In [20] the WMS (Workforce Management System) is described. The main feature of WMS is that it avoids a full search and uses heuristics to produce the second best solution that has been reported so far, in terms of the cost function, and the best solution if we also consider the execution time. It was implemented in CHIP and tested under Andorra-I [18], a CHIP-like system. The key features of the labelling strategy used in WMS are: a) The most constrained jobs are considered first, b) the order of the candidate engineers to do a job is controlled by three different methods/ heuristics [20]:

- **The Closest First:** A technician whose base is closest to the job is selected. This aims to reduce the travel cost. The drawback is that, even if the engineer's base is closer to the job under consideration, it does not mean that the travel time will be the minimum. This is because the technician might be already on tour. The next heuristic compensates for that problem.
- **The Same Direction First:** When a job is going to be labelled, the candidate engineers are organised into two groups. Those that are in a *good direction* related to the job and those that are in a *bad direction*. An engineer belongs to the good direction group if any

of the following conditions hold: a) the engineer is currently idle, b) the engineer has been assigned a job that is in the *same direction* as the labelling job, and c) the engineer has already been assigned more than one job, among which are at least two jobs in the same direction as the labelling job. Two jobs, located at places J1 and J2, are said to be in the *same direction* if the angle `(J1,Base,J2)` is less than 45 degrees.

- **The Least Busy First:** In this heuristic the least busy engineer is selected first. It does not contribute to the reduction of the travelling time but it tries to improve the workforce utilisation.

One last feature of WMS, worth mentioning, is that the complexity of the forward checking algorithm for the *closest* heuristic is *O(n*m)* where *n* is the total number of jobs and *m* is the average length of all engineers' job lists. This linearity allows the use of WMS with larger datasets [20].

4. Distributed WMS

4.1 Clustering

The idea behind clustering is to partition the problem (data and search space) into sub-problems of less complexity. As far as these sub-problems are independent, partitioning reduces dramatically the execution time needed to solve them, as well. This time saving allows us to use algorithms that are impractical in the case of the whole problem due to their complexity. The overhead is that, when the sub-problems are related with constraints on elements belonging to different sub-problems, additional effort is required to resolve conflicts.

We decided to use a clustering method that shares the service demand (i.e. the jobs) among bases in a way proportional to the ratio of the total service capacity of each base. We did not consider the various job types because jobs of the same type are uniformly distributed over the problem's area. This large grained partitioning reflects the natural organisation of BT's services that are also base-oriented.

Another fact is that the *dynamic workforce quality distribution* is also uniform among bases. This term refers to the availability of technicians in a base, to serve jobs that are around that base. Although it is expected to vary from day to day, because the available jobs are different every day, we believe that this variation will be minor; the fact that the dataset is real supports that belief. This eliminates the need to take into account the *job's engineer list* during clustering. Three clustering methods were implemented, namely *balloon*, *center of gravity* and *mixed*.

The balloon clustering method is distributed with centralised control (i.e. one base/agent has the control at any time). The algorithm is given below:

1. Each base creates its *input pool* that contains all the jobs and is sorted in ascending order of their distance from that base.
2. The *virtual service capacity* (VSC) for each base is calculated according to the following function:

$$VSC_i = \frac{RealServiceCapacity_i}{\sum_{j=1}^{Bases} RealServiceCapacity_j} * TotalServiceDemand * 2$$

   where: *RealServiceCapacity*$_j$ is the total free time of all the engineers that belong to the base *j*, *TotalServiceDemand* is the total duration of all jobs, and *Bases* is the number of bases. Thus, the virtual service capacity of base *i* is proportional to the ratio of the total service capacity of that base.
3. The sharing rate for a base *i* is defined as follows:

$$SharingRate_i = \begin{cases} 2 & TechsOfBase_i = 1 \\ 2*ceiling(\ln(TechsOfBase_i)) & TechsOfBase_i > 1 \end{cases}$$

where: *ceiling(X)* denotes the rounded up value of *X*.

4. The sharing order is defined in terms of the initial VSC values, (i.e. the base with the smallest VSC will get jobs first), and is fixed during clustering.
5. While there are still jobs left to share, the base that has the control (we name it *i*) does:

```
if SharingRate_i=0 {
        reset SharingRate_i to its initial value (the one calculated in step 3)
        give the control to the next base in the sharing order }
else if (VSC_i<0 AND JobsInOutputPool_i<6) OR (VSC_i>0) {
                get the first job (named J) from the InputPool_i and put it in the OutputPool_i
                reduce the SharingRate_i by one
                reduce the VSC_i by the duration of job J
                send a message to the rest bases to remove job J from their InputPool }
else give the control to the next base in the sharing order.
```

This method is applied in cycles and terminates when all jobs have been assigned to a base. This is guaranteed to happen because $\sum_{i=1}^{Bases} VSC_i > DemandCapacity$. It is easy to prove that the complexity of the clustering algorithm is *O(NumberOfJobs^2)*. Note that, if the base that has the control is the last in the sharing order, then the control is passed to the first base.

In the center of gravity clustering method each job is selected by the base that is closer to that job as long as the following condition holds: (VSC_i<0 AND $JobsInOutputPool_i$<6) OR (VSC_i>0). The order according to which the jobs are considered for sharing is defined by their distance (travel time) from the *center of gravity* , starting from the jobs that are farther from that point. The center of gravity is the location on the problem's area (∗ point in Figure 2-right) which is the center of gravity of the *virtual multilateral* (dotted line in Figure 2-right) defined by the locations of the bases. Each corner of this multilateral is assumed to have weight equal to the number of the personnel of that base. This clustering method is purely distributed, that is each base works independently.

Note that, in both methods, the minimum number of jobs for every base is 6 and is explicitly defined in the algorithms. This was introduced to make the construction of the unique tour, for a base with a single engineer, more flexible.

	B1	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**	**B8**	**B9**	**B10**	**B11**	**Sum**
Technicians	1	10	1	34	15	1	1	52	1	1	1	118
Jobs-Balloon cluster.	6	36	6	62	45	6	7	64	6	6	6	250
Jobs-Mixed clustering	6	33	6	80	51	6	7	43	6	6	6	250

Table 1: Distribution of jobs to bases (clusters) for the two clustering methods used.

The results of the balloon and center of gravity clustering methods are given in Table 1 and visualised in Figure 2. To evaluate these results we also visualised the solution obtained by all three heuristics used in WMS [20]. The outcome was that each base dominates its surrounding area, with some deviations for some small bases. We derived a visual post-clustering for the results in [20] (i.e. a clustering that is a consequence of a solution given by the global approaches used in WMS), and the overall picture was very similar to the result of the balloon method. This strongly suggests that the first solution of the distributed method will be of quality similar to that of the global approach. Furthermore, if we compare the complexity of the global approach with that of the most complex sub-problem (base/agent 8), we should expect, for the case of the *closest* heuristic, a speed up of about 6 for our distributed approach.

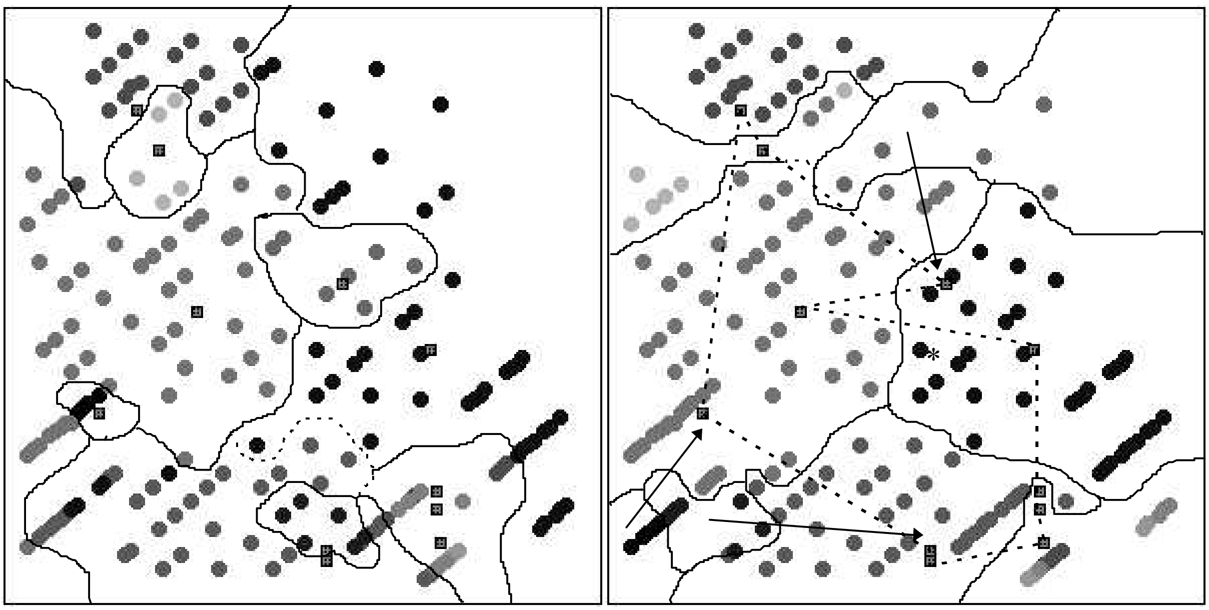
Figure 2: Results of the balloon (left) and center of gravity (right) clustering methods. The clusters for the bases at the lower right corner have been joined, for the purpose of this figure. Both the center of gravity (∗) and the virtual multilateral (dotted line) are also displayed on the right.

The mixed clustering method was introduced to alleviate the assignment of "far" jobs to the small bases that occurred in the center of gravity method (Figure 2 right). According to the mixed method, the small bases select jobs first using the balloon method and then, the remaining jobs are assigned to the rest bases according to the center of gravity method. Since this mixed method seemed more promising than the center of gravity, the latter was rejected. The visualisation of the mixed clustering gave an overall picture similar to the one of the balloon method.

4.2 First Solution Evaluation

A complete WMS system was attached to each agent to solve each sub-problem. A D-WMS-Agent consists of the following modules: The **clustering module**, the **WMS-variant module** which creates the first solution and is almost the same as the original WMS and the **optimiser module** which optimises the first solution and is described later. A block diagram of the architecture of the D-WMS is given in Figure 3. Note that, unlike the clustering and optimisation phases, sub-problem solving is performed without any co-operation between the solving agents.

Table 2 contains a comparison between the global approach (WMS) and the first solution of the distributed one (D-WMS). The first thing worth mentioning is that the first solution is quite good in terms of scheduled jobs. A total of 165 jobs were allocated to engineers using either *closest (H1)* or *same direction (H2)* heuristic for the case of balloon clustering. These numbers are not as good as those in WMS but this was expected, since the D-WMS agents had less jobs and technicians to select from, during the labelling phase. The results for the mixed clustering where a bit worse than those of the balloon clustering, in both heuristics. This was clearly a result of the assignment of less jobs (43 instead of 64) to the base with the most technicians (base 8 - 52 technicians).

The execution time of D-WMS for that stage was the one of the slowest agent. This was the agent 4 for the *closest* heuristic, while for the *same direction* heuristic was agent 8 in both clustering methods. Thus we had a speed up of about 5 using the *closest* heuristic. We recall that, based on the complexity of the forward checking algorithm of WMS and using the most complex sub-problem (cluster 8), we had predicted a speed up of about 6 for the sub-problem solving task. The speed up for the *same direction* heuristic was about 2.5.

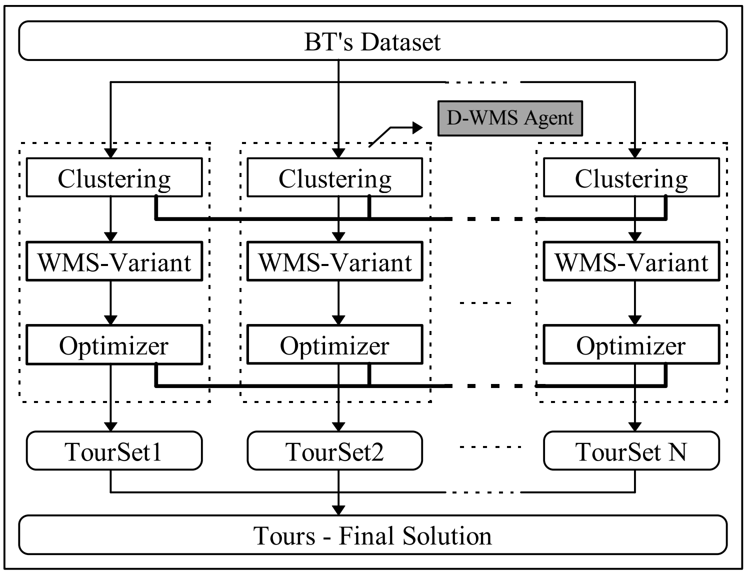

Figure 3: Block diagram of the D-WMS.

Another important issue is that, the total number of active engineers (= number of tours) in D-WMS is less than the corresponding number in WMS, in all cases (Table 2). This, together with the greater number of unscheduled jobs in D-WMS, strongly suggests that it is possible to assign unscheduled jobs of a cluster to technicians belonging to a different base/cluster.

	WMS		**D-WMS**			
Clustering Method	**no clustering**		**Balloon Clustering**		**Mixed Clustering**	
Heuristic used	**H1**	**H2**	**H1**	**H2**	**H1**	**H2**
Scheduled Jobs	191	196	165	165	158	162
Active Techs (=Tours)	67	68	61	62	59	60
Total Cost	23563	21426	26534	24976	27736	25734
CPU Time (sec)	10.7	22.7	2.2	9.3	2.1	8.5

Table 2: Comparison between WMS and first solution of D-WMS. H1 and H2 stand for the *closest* and *same direction* heuristics respectively.

Finally, if we ignore the execution time, it is possible to obtain a better overall first solution by using in each agent the heuristic that gives the best local solution. This is possible because the sub-problem solving step is purely distributed but the disadvantage is that further analysis of the data of each cluster is required to be able to guess the right heuristic.

4.3 Optimisation

Various *repair optimisation methods* have been used in the past in such scheduling problems. The main drawback of them is that they are computationally expensive and they aim mainly to replace jobs in tours with other jobs so as to minimise transitions between jobs and spend more time on actual work, minimising the cost function in that way. Such a method implemented in a distributed environment will become even more complex because it requires a lot of communication between the solving agents.

In the first solution of D-WMS we agree that there are cases of tours which include long transitions and are promising candidates for a repair method. But we believe that we can, at least, reach the quality of the solution of WMS with less effort. This is mainly because during the generation of the first solution, each agent/base was not aware of the jobs of the neighbouring bases. If we inform each agent for the existence of the unscheduled jobs of the other bases/agents, then we should expect that some of them could be used to fill in periods in which the technicians are idle. This is demonstrated in Figure 4 where we visualise some tours of the first solution of the *balloon/closest* approach. In this figure, we can see tours with a lot of used time (e.g. 1, 9, 10) as well as tours with a lot of unused time (e.g. 11, 41, 42). This free time together with the free time of the idle technicians we tried to exploit to improve the tours of the first solution, using a distributed and computationally cheap method.

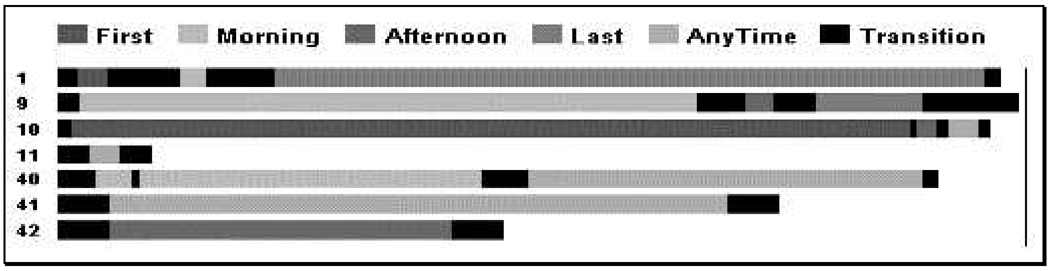

Figure 4: Visualisation of some tours of the first solution of the balloon/closest approach. Each bar represents the tour of the technician with the ID displayed on the left. The vertical line on the right denotes the end of the 8 hour working day. It is obvious that there are tours with enough free time to try to assign to them more jobs.

We have developed a *patching method* instead of a repair one. This is distributed, involves little communication between the agents, and can be applied in cycles until no further optimisation is possible. It consists of identifying possible gaps in a tour that can be filled (i.e. convert gaps to patches) with one of the unscheduled jobs. Two sources of improvement are possible:

- Unscheduled jobs belonging to an agent/cluster could be assigned (skill constraints permitted) to active technicians who have enough free time and belong to another cluster/ base, and
- Unscheduled jobs belonging to an agent/cluster could be assigned (skill constraints permitted) to idle technicians of another cluster/base.

```
During an optimisation cycle $\mathtt{Agent}_{i}$ does:
1. sends/receives the unscheduled jobs to/from the other agents
2. for each $\mathtt{Tech}_{ij}$ generates all $\mathtt{Gap}_{ijk}$ permutations based on the current tour $\mathtt{Tour}_{ij}$ of $\mathtt{Tech}_{ij}$
3. for each $\mathtt{Gap}_{ijk}$ generates all possible patches (named $\mathtt{Patch}_{ijkl}$) using jobs from the set of the
   unscheduled jobs
4. sends/receives all patches $\mathtt{Patch}_{ijkl}$ to/from the other agents
5. filters out the whole set of patches in an effort to keep the ones that further improve the first solution
   and to resolve possible conflicts (i.e. prevent the use of a job in more than one patches)
6. uses the remaining of the patches $\mathtt{Patch}_{ijkl}$ to update (to patch) the tours $\mathtt{Tour}_{ij}$
```

Figure 5: An outline of the optimisation algorithm used by each D-WMS Agent to improve the first solution. Steps 1 and 4 involve only exchange of information between agents. Steps 2, 3, 5 and 6 are Gap Generation, Patch Generation, Patch Filtering and Patching, respectively.

An outline of the optimisation algorithm is illustrated in Figure 5. After the first solution has been produced, each agent sends its unscheduled jobs to the other agents, and receives from them their unscheduled jobs (step 1 in Figure 5). This is the first point during the optimisation phase that involves communication between the agents. Then, each agent identifies possible gaps (step 2) in its technicians' working timetable that can be filled, in step 3, with one of the unscheduled jobs (i.e. convert gaps to patches).

A second exchange of information between agents takes place in step 4. This time the piece of information exchanged is the patches generated in step 3. Finally, in step 5, each agent locally filters out the set of patches to resolve possible internal (between engineers) or external (between agents) conflicts. Those of the remaining patches that refer to tours of that particular agent (named Agent_i), are used to improve the first solution produced by that agent (step 6). The rest patches refer to first solution tours created by other agents, and although are ignored by Agent_i, they are used by the other agents.

The above steps outline one cycle of the optimisation loop. Although more than one gaps (and patches) are defined for each tour, only one is used, at maximum, to improve a tour. The same algorithm can be used again to further optimise the tours, if possible. If this is not true, then the optimisation phase terminates.

4.3.1 Gap Generation

According to the current state of an engineer, two different algorithms are used by each agent to create the gaps. These are:

- *Gap Generation for idle engineers*
  Because there is no real tour for an idle technician, we introduce the *null tour* which starts from and ends at a certain base and serves no jobs. Two gaps are created from a null tour: a) an eight hour (`640` minutes) gap that can be filled with a job of any of the types [first, last, morning, any_time]. Jobs of type "afternoon" were excluded, because are mutually exclusive with "morning" jobs, and b) a gap that can be filled with jobs of type [first, last, afternoon, any_time] and lasts from the midday to the end of the working day of the engineer.
- *Gap Generation for active engineers*
  For any given tour that serves `N` jobs there are `N+1` potential locations between two successive stop points in the tour. Each of these locations, under certain conditions, can be used to locate a gap. Note that some locations are immediately rejected since they are ruled by hard constraints. For example, it is not possible to serve any new job between a last job and the arrival to a base at the end of an existing tour. The actual duration of a generated gap between two existing stop points P1 and P2 is adjusted on the fly to include the current transition duration from P1 to P2, and is augmented with the valid types of candidate jobs. For example, if a gap starts after midday, then it cannot be filled in by an unscheduled job of type "morning".

4.3.2 Patch Generation

Each gap_{ijk} instance generated in the previous step can be used to produce patches. A *patch* is a gap instance with the following properties:

- Its free time is fully or partially covered by the duration of one of the unscheduled jobs plus the transitions to and from this job.
- The type of the job used is one of the valid job types for that gap.
- Modification of the tour the patch belongs to, reduces the cost function by a certain amount called `Gain`.

It is clear that, the number of patches that will be produced by using one gap, depends mainly on the number and the attributes of the unscheduled jobs. Moreover, any of the unscheduled jobs can be used in more than one patches. In other words, the set of patches created by each agent is not consistent neither internally nor externally (between the patch sets of various agents). These conflicts will be resolved in the next step of optimisation (Patch Filtering), that takes place after all agents exchange their own patch set, between each other.

4.3.3 Patch Filtering

The patch filtering step is performed locally, in each agent, but is made based on the global patch set. As a result, redundant work is performed by each agent trying to filter out the global patch set. The advantage of this is that no further communication is necessary. This is because both the filtering algorithm and the dataset to filter are exactly the same for every agent. As a result the filtered set of patches is the same for every agent, but each agent will keep at the end only the patches that refer to its own technicians.

The patch filtering algorithm used by every agent is a three step algorithm and is applied continuously (in loops) until the global set of patches becomes empty. The result is a set of "good patches" that will be used later to optimise the tours. The patch filtering algorithm is described in Figure 6.

```
   Assuming that, after each optimisation loop there are N different tours for which there exist entries
   into the global set of patches, each agent does:
1. Creates N groups each one containing all the patches of a single tour and sorts each of them on the
   Gain value in descending order.
2. Creates the priority list which defines from which tour (group of patches) the filtering should start.
   The criteria used to assign priorities are the number of patches in each group, the gain value of the
   very first patch in each group and the tour index (ID), in this order.
3. Selects the tour (group of patches) with the highest priority:
   if (it has only one patch) then {
          discards all the patches that use the same job from any other group
          puts this patch into the "good patches" pool }
   else { applies the LookAhead mechanism to define which patch to use
          discards all the patches that use the same job, from any other group
          discards the remaining patches of that group (i.e. those refer to the same tour)
          puts this patch into the "good patches" pool }
```

Figure 6: The patch filtering algorithm.

Aiming at a maximum reduction into the cost function during optimisation, we have developed a *Look Ahead* mechanism which works as follows: When the tour under consideration has more than one patch and the first one is also the first/best patch of another tour (which is later in the priority list) then the agent also examines the gain of the second patch. For example, say that tour T1 is under consideration and its first two patches are J1/G1 and J2/G2 (JobID and Gain values respectively). If there is another tour T2, with best (first) patch J1/G3 (i.e. uses the same job with the best patch of tour T1) and 2nd patch J4/G4 then:

```
if (G1+G4 ≥ G2+G3) then (agent uses patch J1/G1 for tour T1)
else agent does{
    ignores patch J1/G1 from the patches of T1 (that is, leaves J1/G1 for tour T2)
    tries in a similar way to figure out if J2/G2 is the "good patch" for tour T1}.
```

When the patch filtering is over, each agent updates only its own tours used those of the "good patches" that refer to local tours. Tour updating is done by inserting the job of a patch into the right position of the job list of the tour.

4.3.4 Final Solution Evaluation

Apart from the quality, there is no difference between the first solution and the optimised tour sets. The consequence is that we can apply the optimisation algorithm successively until no further optimisation is possible, a case in which all generated patches are filtered out immediately. We should expect further optimisation for two reasons: a) Some of the candidate tours for optimisation in the first optimisation cycle were not optimised at all, because their patches were discarded during patch filtering, and b) It is very likely that, among the optimised tours there are some that still have enough free time to serve a job. This "no further optimisation" situation happened on the third optimisation cycle for all clustering/heuristic combinations. The most important figures of the total solution after each cycle are given in Table 3.

	Final Solution			
Clustering Method	**Balloon**		**Mixed**	
Heuristic used	**H1**	**H2**	**H1**	**H2**
Scheduled jobs	185 (165+18+2)	187 (165+20+2)	178 (158+17+3)	181 (162+16+3)
Active Techs	67 (61+6+0)	67 (62+5+0)	65 (59+6+0)	64 (60+4+0)
Total Cost	**23775** (26534)	**22078** (24976)	25099 (27736)	23202 (25734)
Total Opt/tion time	5.4 sec	5.9 sec	6 sec	5.5 sec

Table 3: Final solution obtained by D-WMS after three optimisation cycles. The numbers in brackets refer to the contribution to the final solution of the first solution, the first and the second optimisation cycles respectively.

First of all, it is clear that the balloon clustering method gave better results than the mixed one. The total cost we achieved for the closest (same distance) heuristic was 23775 (22078), a value very close to the corresponding results of the WMS in [20]. During the optimisation phase, 20 (22) more jobs were served resulting in a total of 185 (187) scheduled jobs against 189 (191) of WMS. This is a very good performance of D-WMS because, although slightly worse at first glance in terms of scheduled jobs and total cost, it was achieved in shorter time. In Figure 7 we have visualised the final solution of D-WMS (left) for the *balloon*/*same direction* approach, as well as, the solution of WMS for the same heuristic. The similarity of the tour patterns in those figures is remarkable.

The execution time of the optimisation phase is also given in Table 3. The most complex tasks of the optimisation are the gap and patch generation. We estimated the order of complexity of the "gap and patch generation" steps to be:

$$O((ScheduledJobs + 8*IdleTechs)*UnscheduledJobs)$$

It is obvious that, the more the scheduled jobs the fewer the unscheduled jobs. In addition, more scheduled jobs generally implies less idle technicians. This explains why the "gap and patch generation" task duration was almost the same for every optimisation cycle. The "patch filtering" and "tour update" tasks were quite fast This was because after a patch is selected as a "good patch", a lot of pruning is performed by discarding all patches that use the same job with the "good patch". In addition, all patches that refer to the same tour with the selected patch are also discarded.

In overall, for the balloon clustering method, we had the following results for the *closest* (*same direction*) heuristics: 3% (4.8%) worse performance in number of scheduled jobs, 0.01% (0.03%) worse performance in terms of the cost function and 40% (50%) better performance in terms of execution time.

Since our optimisation algorithm does not only optimise existing tours but also creates new tours by assigning unscheduled jobs to idle engineers, we applied it to the empty tour set, that is, we assumed that all the technicians are idle and all the jobs unscheduled. This process was performed by a single agent and it took five cycles and about 3 minutes to complete. The result was 105 tours, 177 scheduled jobs and a total cost of 25660. This "from scratch" tour generation gave a tour set that when visualised revealed the nature of the optimisation algorithm which is very close to the *less busy* heuristic used in WMS.

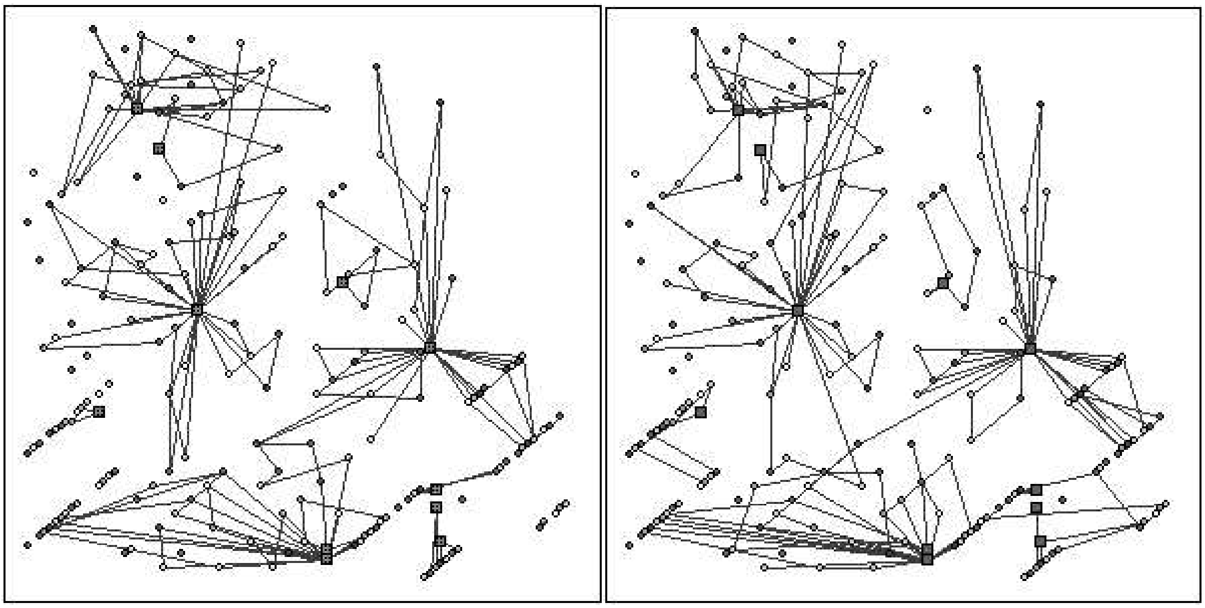
Figure 7: Final solution obtained by D-WMS (left) and WMS (right) using the *closest* heuristic.

Finally, we applied the optimisation algorithm to the results obtained by WMS in [20]. The result was two more allocated jobs and a small reduction to the total cost.

5. Related Work

This particular problem of BT has been deeply studied in the past by using various centralised approaches: the genetic algorithm approach [15], simulated annealing [2], a guided local search approach [19] and two CLP based approaches [1, 20]. We have selected the best results from each approach and listed them in Table 4. Apart from [20], the results published so far are all refer to the BT relaxed DT-250-118 dataset, in which all "first" jobs are assumed "morning" jobs, and all "last" jobs are assumed "afternoon" jobs. Our results refer to the original dataset and could be directly compared to the corresponding case of [20].

The guided local search (GLS) [19] is a sophisticated algorithm designed to overcome the hill climbing's local optima problem. Table 4 shows that it has produced the best result on this set of data at a rather high cost (execution time). No timing was reported for the genetic algorithm approach [15] or simulated annealing [2]. A repairing technique was applied in [1] and [15] to optimise the unallocated jobs further. This is not used in [20]. The CLP based approach from BT [2] has a few similarities with [20], however, the forward checking proposed in [20] is not used in [2].

We remind that all the above approaches are centralised, i.e. a single solving process is performing all the computation. An agent based approach to CSPs has been studied in [8] and [4]. In [8] the MARS system is introduced which is a generic approach to implement a testbed that allows a CSP to be modelled as a community of co-operative agents. This work focuses primarily on the agency aspects of the approach and is applied to transportation scheduling.

Furthermore, techniques for dynamic replanning are introduced, within the same framework. Similar approaches are followed in [4] were the CHRONOS multi-agent environment is described and is used to solve a teaching space utilisation problem. Our agent based method, although is not as generic as the above two, is far less comlex and for the case of naturally distributed CSPs we believe that it outperforms both of them in situations where quasi-real time performance is required, since it does not rely on complex agent-to-agent communication protocols.

Approaches	**Total Cost**	**Travel Cost**	**Unallocated Cost (Jobs)**	**CPU Time (sec)**	**Hardware**
Relaxed Dataset					
GA *	22570	NA	NA (54)	NA	
SA	21050	4390	16660 (56)	NA	
GLS	20433	4707	15726 (48)	9183	DEC Alpha, C++
CLP: BT *	21292	4902	16390 (53)	600	
CLP: WMS	20981	4716	16220 (54)	97	SUN-Sparc20, Andorra-I
Original Dataset					
CLP: WMS	21426	4800	16626 (54)	95	SUN-Sparc20, Andorra-I
CLP: WMS	21426	4800	16626 (54)	23	SUN-UltraSparc, Andorra I.
CLP: D-WMS **	22078	4731	17347 (63)	15	SUN-UltraSparc, Andorra-I, SICStus
*: after repair **: after patching					

Table 4: Comparison of different approaches applied to the BT DT-250-118 dataset.

6. Conclusions and Future Work

We presented a distributed (multi-agent oriented) approach to the workforce management problem. First, we evaluated the results obtained by a Constraint Logic Programming based system (WMS) developed in the past [20], and then, based on this evaluation we proposed a distributed version of it, namely the D-WMS. We have show that certain types of scheduling problems that are naturally distributed can be tackled efficiently, in terms of solution quality and execution time, by adapting existing global approaches (like the WMS) in a distributed environment.

The solving unit of D-WMS is the D-WMS Agent. It is consisted of three main modules, each one responsible for one of the main tasks of our approach. These are:

- the co-operative partitioning of the problem,
- the independent solving of each sub-problem to get a first solution, and
- the co-operative improvement of the first solution.

We utilised the WMS in our agents as it was (with no modification) because, on the one hand, the WMS had performed very well in [20] and from the other the natural distribution of the problem was a guarantee that we will have an adequate first solution to try to improve. In the first and third tasks, which involved co-operation between agents, we had to keep the communication between them as low as possible.

The results were very encouraging. The techniques proposed performed well on the set of real data. Not only was the final solution of good quality, but also the communication between the agents was limited to a minimum level, achieving a short execution time. Furthermore, all of the algorithms scale up well, thus they can be used with larger datasets. According to [20] the same holds for the WMS. Moreover, the most complex of our algorithms can be massively parallelised with no extra cost, resulting in greater speed ups. For example, the generation of gaps for a tour could be a single independent process. The same holds for the patch generation step. Th

D-WMS was simulated on a UltraSparc (175 MHz) based SUN machine using SICStus and Andorra I Prolog (the sequential version). The visualisation tools where implemented in LPA-Prolog for Windows. The D-WMS system is currently ported to a network of UNIX workstation using the Linda facilities.

Acknowledgements

We would like to thank Rong Young for helpful discussions and for providing us the WMS.

References

[1] Azarmi N. and Abdul-Hameed W., *Workforce scheduling with constraint logic programming.* British Telecom Technology Journal, Vol.13, No.1, British Telecom Laboratories, Ipswich, (1995).

[2] Baker S., *Applying Simulated Annealing to the Workforce Management Problem.* Technical Report, British Telecom Laboratories, Ipswich, (1993).

[3] British Telecom Laboratories, RD-250-118 data set of WMS problem, with the permission of British Telecom "http://cswww.essex.ac.uk/CSP/wfs"

[4] Das S.K., El-Kholy A., Harrison C.A., Liatsos V. and Richards E.B., *A multi-agent view of planning and scheduling,* in Proceedings of the Expert Systems 1995 Conference, pp.361-367, (1995)

[5] Dechter R. and Pearl J., *A problem simplification approach that generates heuristics for constraint satisfaction problems*. In Hayes J.E., Michie D., and Richards J. (Eds.), Machine Intelligence, Oxford: Clarendon Press, (1987), pp.125-155

[6] Dechter R. and Pearl J.,*Tree clustering for constraint networks*. Artificial Intelligence, 38, (1989), pp.353-366

[7] Dechter R., *Enhancement schemes for constraint processing: backjumping, learning and cutset decomposition*. Artificial Intelligence, 41, (1990), pp.273-312

[8] Fischer K., Muller J.P. and Pischer M., *Cooperative Transportation Scheduling: an Application Domain for DAI,* DFKI Research Report RR-95-01, (1995)

[9] Haralick A.M. and Elliot G.L., *Increasing tree search efficiency for constraint satisfaction problems*. Artificial Intelligence, 14(3), (1980), pp.263-313

[10] Liu J. and Sycara K.P., *Distributed constraint satisfaction through constraint partition and co-ordinated reaction*. In 12th Distributed Artificial Intelligence Workshop, Hidden Valley, Pennsylvania, (1993), pp.263-279

[11] Liu J. and Sycara K.P., *Distributed problem solving through co-ordination in a society of agents*. In Proceedings of the 13th Distributed Artificial Intelligence Workshop, Seattle, Washington, (1994), pp.190-206

[12] Luo Q.Y., Henry P.G. and Buchanan J.T., *Strategies for distributed constraint*

satisfaction problems. In proceedings of the 13th Distributed Artificial Intelligence Workshop, Seattle, Washington, (1994), pp.207-221

[13] Mackworth A.K. and Freuder E.C., *The complexity of some polynomial network consistency algorithms for constraint satisfaction problems*. Artificial Intelligence, 25, 1985, pp.65-74

[14] Montanari U., *Networks of constraints: fundamental properties and applications to picture processing*. Information Sciences, 7, (1974), pp. 95-132

[15] Muller C., Magill E.H. and Smith D.G., *Distributed Genetic Algorithms for Resource Allocation.* Technical Report, Strathclyde University, Glasgow, (1993).

[16] Nishibe Y., Kuwabara K. and Ishida T., *Effects of heuristic in distributed constraint satisfaction: towards satisfying algorithms*. In Working papers of the 11th International Workshop on Distributed Artificial Intelligence. Glen Arbor, Michigan, (1992), pp.285-302

[17] Purdom P.W.Jr, *Search rearrangement backtracking and polynomial average time*, Artificial Intelligence, 21, (1983), pp.117-133

[18] Santos Costa V., Warren D.H.D. and Yang R., *Andorra-I: a parallel Prolog system that transparently exploits both and- and or- parallelism*. In Proceed. of the 3rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, (1991).

[19] Tsang, E.P.K. and Voudouris, C., *Fast local search and guided local search and their application to British Telecom's workforce scheduling problem*, Operations Research Letters, Elsevier Science Publishers, Amsterdam, Vol.20, No.3, (1997), 119-127

[20] Yang R., *Solving the Workforce Management Problem with Constraint Programming.* Proceedings of Practical Applications of Constraint Technology, (1996).

[21] Yokoo M., *Asynchronous Weak-commitment Search for Solving Distributed Constraint Satisfaction Problems.* Proceedings of the 1st International Conference on Constraint Programming, (1995), 88-102.

[22] Yokoo M., Durfee E.H., Ishida T. and Kuwabara K., *Distributed Constraint Satisfaction for Formalising Distributed Problem Solving.* Proceedings of the 12th IEEE International Conference on Distributed Computing Systems, (1992), 614-621.

[23] Yokoo M., *Dynamic value/variable ordering heuristics for solving large scale distributed constraint satisfaction problems*. In Proceedings of the 12th Distributed Artificial Intelligence Workshop, Hidden Valley, Pennsylvania, (1993), pp.407-422

[24] Yokoo M., Ishida T. and Kuwabara K., *Distributed constraint satisfaction for DAI problems*. In Proceedings of the 10th International Workshop on Distributed Artificial Intelligence, Bandera, Texas, (1990)

[25] Zhang Y. and Mackworth A., *Parallel and distributed algorithms for finite constraint satisfaction problems.* Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, (1991), pp.394-397.