Dynamic Agent Prioritisation with Penalties in Distributed Local Search

Amina Sambo-Magaji, Inés Arana and Hatem Ahriz
School of Computing Science and Digital Media, Robert Gordon University, Aberdeen, U.K.

Keywords: Distributed Problems Solving, Local Search, Distributed Constraint Satisfaction, Heuristics.

Abstract: Distributed Constraint Satisfaction Problems (DisCSPs) solving techniques solve problems which are distributed over a number of agents.The distribution of the problem is required due to privacy, security or cost issues and, therefore centralised problem solving is inappropriate. Distributed local search is a framework that solves large combinatorial and optimization problems. For large problems it is often faster than distributed systematic search methods. However, local search techniques are unable to detect unsolvability and have the propensity of getting stuck at local optima. Several strategies such as weights on constraints, penalties on values and probability have been used to escape local optima. In this paper, we present an approach for escaping local optima called Dynamic Agent Prioritisation and Penalties (DynAPP) which combines penalties on variable values and dynamic variable prioritisation for the resolution of distributed constraint satisfaction problems. Empirical evaluation with instances of random, meeting scheduling and graph colouring problems have shown that this approach solved more problems in less time at the phase transition when compared with some state of the art algorithms. Further evaluation of the DynAPP approach on iteration-bounded optimisation problems showed that DynAPP is competitive.

1 INTRODUCTION

Constraint Satisfaction Problems (CSPs) are problems which can be expressed by a set of variables, the set of possible values each variable can be assigned and a set of constraints that restrict the values variables can be assigned simultaneously (Dechter, 2003). Distributed Constraint Satisfaction Problems (DisCSPs)(Yokoo et al., 1998) are CSPs where the problem is dispersed over a number of agents which have to cooperate to solve the problem while having limited knowledge about the overall problem. It is often assumed that each agent is responsible for the assignment of one variable only and this will be also assumed in this paper. A solution to a DisCSP is found when an assignment of values to all the variables is found which satisfies all existing constraints (Rossi et al., 2006).

Systematic (backtracking) and local (iterative) search are two types of approaches for solving DisCSPs. Systematic algorithms are complete but slow for large problems when compared to local search, which is incomplete. Local search algorithms start with an initial random instantiation of values to all the variables and, in successive iterations, these values are changed to repeatedly reduce the number of constraint violations until a solution is found. Local search has the propensity of getting stuck at local optima - a point where the currently proposed assignment (that is not a solution) cannot be further improved by changing the value of any single variable. A Quasi Local Optima is a weaker form of local optima in local search where an "unsatisfied" agent (with at least one constraint violation) and all its neighbours have no single improvement on their current assignment (Zhang et al., 2002).

In this paper, we present Dynamic Agent Prioritisation and Penalties (DynAPP) a new approach for escaping local optima which combines dynamic variable prioritisation with penalties on variable values for escaping local optima hence, avoiding search stagnation. An empirical evaluation with instances of random, meeting scheduling and graph colouring problems showed that, at the phase transition, DynAPP solved more problems in less time compared with state of the art algorithms. Evaluation of DynAPP on iteration-bounded optimisation problems showed that DynAPP is competitive.

The remainder of this document is structured as follows: section 2 describes related work; DynAPP is explained in section 3; an empirical evaluation of DynAPP is presented in section 4; finally, conclusions are drawn section 5.

Sambo-Magaji A., Arana I. and Ahriz H..
Dynamic Agent Prioritisation with Penalties in Distributed Local Search.
DOI: 10.5220/0004259202760281
In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence* (ICAART-2013), pages 276-281
ISBN: 978-989-8565-38-9
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

2 RELATED WORK

A number of strategies have been implemented to escape and avoid local optima in distributed search which has resulted in several algorithms. The Distributed Breakout Algorithm (DBA)(Yokoo and Hirayama, 1996) is a hill climbing algorithm that increases the weight of violated constraints in order to raise the importance of constraints that are violated at local optima, thus forcing the search to focus on their resolution. The original algorithm has been studied extensively and a number of improved versions proposed. Multi-DB (Hirayama and Yokoo, 2002) is an extension of DBA for Complex Local Problems - i.e. DisCSPs with more than one variable per agent. Multi-DB was then improved in (Eisenberg, 2003) by increasing constraint weights only at global optima with the new algorithm being called DisBO. (Basharu et al., 2007b) proposed DisBO-wd an improvement on DisBO where constraint weights are decayed over time. Thus, at each step, the constraint weight is increased if the constraint is violated and it is decayed if the constraint is satisfied. SingleDBWD (Lee, 2010) is a version of DisBO-wd for DisCSP with one variable per agent.

Another DisCSP algorithm is Distributed Stochastic algorithm (DSA)(Zhang et al., 2002), a randomized local search algorithm that uses probability, to decide whether to maintain its current assignment or to change its value at local optima. A hybrid of DSA and DBA was proposed in Distributed Probabilistic Protocol (DPP) (Smith and Mailler, 2010) where the weights of constraints violated at local optima are increased and agents find better assignments by the use of probability distributions.

The Stochastic Distributed Penalty Driven Search(StochDisPeL) (Basharu et al., 2006) is an iterative search algorithm for solving DisCSPs where agents escape local optima by modifying the cost landscape by imposing penalties on domain values. Whenever an agent detects a deadlock (quasi-local optima), StochDisPeL either imposes a temporary penalty (with probability p) to perturb the solution or increases the incremental penalty (with probability $1-p$) to learn about the bad value combination. Incremental penalties are small and remain imposed until they are reset while temporary penalties are discarded immediately after they are used. The penalties on values approach has been shown to outperform the weights on constraints approach of escaping local optima (Basharu et al., 2007a).

Asynchronous Weak Commitment Search (AWCS) (Yokoo, 1995) is a complete asynchronous backtracking algorithm that dynamically prioritises agents. An agent searches for values in its domain that satisfy all constraints with higher priority neighbours, and from these values it selects the value that minimises constraint violations with lower priority neighbours. When an agent does not find a consistent assignment, the agent sends messages called no-good to notify its neighbours and then increases its priority by 1. The use of priority changes the importance of satisfaction of an Agent.

3 DynAPP: DYNAMIC AGENT PRIORITISATION AND PENALTIES

We propose Dynamic Agent Prioritisation and Penalties (DynAPP) [Algorithms 1-4] - a new algorithm that combines two strategies: penalties on values and agent prioritisation. At local optima, the priority of inconsistent agents (whose current variable assignment leads to constraint violations) is increased and, at the same time, a diversification of the search is encouraged by penalising values which lead to constraint violations.

In DynAPP, variables are initialised with random values and all agent priority values are set to their agent ID. It should be noted that the lower the agent ID, the higher the actual priority of the agent. An agent then sends its initial variable assignment to its neighbours. Each agent takes turns to update their AgentView (their knowledge of the current variable assignments) with the messages received and selects a value for its variable that minimises the following cost function:

$$c(d_i) = viol(d_i) + p(d_i) \qquad i \in [1..|domain|]$$

where d_i is the ith value in the domain, $viol(d_i)$ is the number of constraints violated if d_i is selected and $p(d_i)$ is the incremental penalty imposed on d_i. When a temporary penalty is imposed, the penalty is used to select another improving value and immediately reset. A QLO is detected when an AgentsView does not change in two consecutive iterations. At QLO, an agent (like StochDisPeL[1]) imposes a temporary penalty (with probability p) or it increases the incremental penalty (with probability $1-p$) and also changes its priority value to the priority of the highest priority neighbour with whom it shares a constraint violation thus elevating itself among its neighbours. The neighbour with the highest priority then reduces

[1]Further details on how penalties are imposed can be found in (Basharu et al., 2006).

its priority by 1. This dynamically reorders the structure of its higher and lower level priority neighbourhood.

Algorithm 1: Dynamic Agent Prioritisation and Penalties (DynAPP): Agent.

1 *random initialisation*
2 penaltyRequest ← null
3 priority ← agentID
4 **repeat**
5 **while** an agent is active **do**
6 *accept messages* and update AgentView and penaltyRequest
7 **if** cost function is distorted **then** reset all incremental penalties **end if**
8 **if** penaltyRequest ! = null **then**
9 **processRequest()**
10 **else**
11 **if** current value is consistent **then**
12 reset all incremental penalties
13 **else**
14 **chooseValue()**
15 **end if**
16 **end if**
17 **sendMessage(penaltyRequest, priority)**
18 **end while**
19 **until** termination condition

Algorithm 2: Procedure processRequest().

1 **for** all the *messages* received
2 **if** penaltyRequest ← IncreaseIncPenalty **then**
3 increase incremental penalty on current value
4 **else**
5 penaltyRequest ← ImposeTemporaryPenalty
6 impose temporary penalty on current value
7 **end if**
8 **end for**
9 select value minimising cost function
10 reset all Temporary penalties

When an incremental penalty is imposed (Algorithm 3 line 10), an agent informs neighbours with lower priority to also increase their incremental penalty on current values; similarly, when a temporary penalty is imposed (Algorithm 3 line 7), an agent requests further temporary penalty impositions on current values to lower priority neighbours with whom it shares a constraint violation. When there is no penaltyRequest, all neighbours are informed of the current value and priority (Algorithm 4 line1,11) . Messages are processed by agents as described in Algorithm 2, increasing or imposing an incremental or temporary penalty respectively. An agent then selects the value minimising the cost function and continues this process until consistent values are found or the maximum number of iterations is reached.

Algorithm 3: Procedure chooseValue().

1 **if** agentView(t) ! = agentView(t-1) **then**
2 select value minimising cost function
3 *return*
4 **end if**
5 r= random value in [0...1]
6 **if** r < p **then**
7 impose **temporary penalty** on current value
8 penaltyRequest ← ImposeTemporaryPenalty
9 **else**
10 increase **incremental penalty** on current value
11 penaltyRequest ← IncreaseIncPenalty
12 **end if**
13 **for** all neighbours violating constraint with currentVar
14 if priority[neighbour] > priority[currentVar]
15 priority[currentVar] = priority[neighbour]
16 end if
17 **end for**
18 priority[neighbour] = priority[neighbour]-1
19 select value minimising cost function

Algorithm 4: Procedure sendMessage(penaltyRequest, priority).

1 **send message**(id, value, priority, null) to all neighbours with > priority
2 **if** penaltyRequest = IncreaseIncPenalty **then**
3 **send message**(id, value, priority, penaltyRequest) to all neighbours
4 with < priority
5 **else if** penaltyRequest = ImposeTemporaryPenalty **then**
6 **send message**(id, value, priority, ImposeTemporaryPenalty) to
7 neighbours with < priority & violating constraints with Self
8 **send message**(id, value, priorityValue, null) to neighbours with <
9 priority not violating constraints with Self
10 **else**
11 **send message**(id, value, priority, null) to neighbours with< priority
12 **end if**

For example, Figure 1 represents the simplistic problem of allocating timeslots for 5 student vivas. Representing this as a DisCSP, A, B, C, D, E are the variables (students) and their domains (timeslots) D_i D_i are D_A= {1,2}, D_B={1,2}, D_C={1,3}, D_D={1,3}, E={2,3}. There are 5 constraints in the problem and V_i ($i \in [A..E]$) represents contraint violations and P_i ($i \in [A..E]$) represents penalties imposed on each domain values. Initially, the priorities for {A,B,C,D,E} are 4,2,3,1,5 respectively, a lower number implies higher priority. Each agents keeps account of its higher priority neighbour (HPN) and lower priority neighbour (LPN). A has no LPN and {B} as HPN, C has {B,D} as HPN and {E} as LPN and so on. If for A=1, B=1, C=1, D=3, E=3, C detects a local optimum, C changes its priority to 2 (i.e. that of B

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

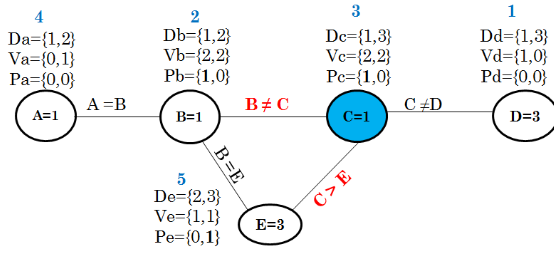

Figure 1: A simple DisCSP.

which is the highest HPN it has a constraint violation with. B reduces its priority to 3. C now has HPN {D} and LPN {B,E}. C then imposes a penalty (assuming an incremental penalty which is 1) on value 1 and informs its new LPN {B,E} to also impose an incremental penatly on their current assignment 1, 3 respectively. Each agent then selects values with lower violation C=3, D=1, E=2, A=2, B=2 which solves the problem.

4 EMPIRICAL EVALUATION

Several problem instances of random problems, graph colouring problems and meeting scheduling problems of were used to compare DynAPP with StochDispel and SingleDBWD. The percentage of problems solved and the number of messages were recorded. The ratio of number of iterations vs constraint violations were evaluated to further verify the algorithm that solves the most problems early and was allowed to run for a maximum 100n iterations, where n is the number of variables,nodes or meetings. Statistical significance was calculated using the wilcoxon test.

Random Problems. We evaluated DynAPP with a variety of random problems with binary constraints and present results (see Figure 2) with the following problem-sizes and densities {80-0.1, 90-0.09,100-0.08, 110-0.07, 120-0.06, 130-0.062, 140-0.057, 150-0.054, 160-0.051, 170-0.047, 180-0.045, 190-0.042, 200-0.04} with a tightness of 0.4 (phase transition) and domain size of 10. Median values for 100 problems are presented which show that DynAPP solved the most problems and had the least number of messages. From 140 variables, SingleDBWD solved less than 40% of the problems. Note: When an algorithm did not solve a problem, the messages "wasted" in that problem were not counted towards the median number of messages.

When evaluating the algorithm by number of iterations vs constraint violations (see Table 1), DynAPP solved some problems as early as at the 50th cycle while StochDisPeL and SingleDBWD had not solved any problem. At the 2000th iteration, with a median constraint violation of 1 for both StochDisPeL and DynAPP, DynAPP solved 84% of the problems as opposed to 63% by StochDisPeL. SingleDBWD solved only 29% of the problems. The differences in performance between DynAPP and the other algorithms is statistically significant.

Table 1: Iterations vs. Violations: Random Problems.

	StochDisPel		DynAPP		SingleDBWD	
N. Itrs	M viols	% sol	M viols	% sol	M viols	% sol
10	16	0	15	0	41	0
50	8	0	8	1	15	0
100	6	0	5	**4**	10	0
500	2	22	3	29	4	0
1000	1	38	2	49	4	2
1500	1	57	1	69	3	15
2000	1	**63**	1	**84**	2	**29**

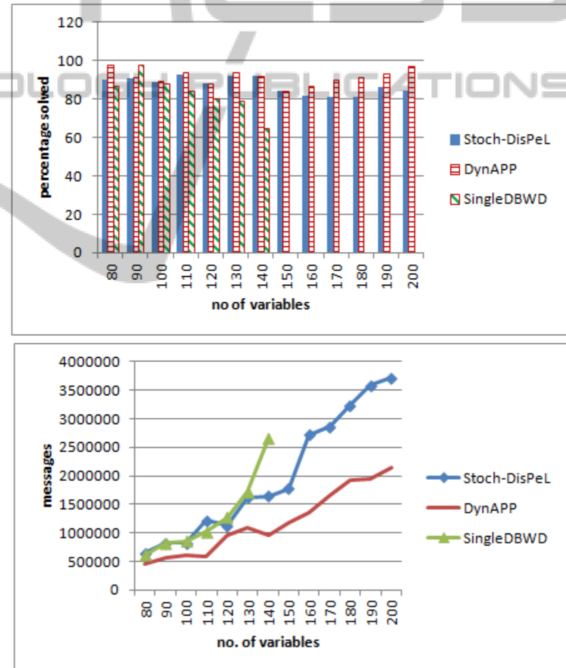

Figure 2: Random Problems.

Graph Colouring. 3-Colour distributed graph colouring problems were solved with n nodes, $n \in$ {100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200} with degree between $4.3 \leq degree \leq 5.3$. The results, shown in Figure 3, are for the different graph sizes at degree 4.9. DynAPP solved the most problems and had the least number of messages. DynAPP solved 3 problems as early as at the 50th cycle while StochDisPel solved 1 and SingleDBWD did not solve any problem as seen in Table 2. At the 2000th iteration, DynAPP solved 93% of the problems as opposed

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

to 68% by StochDisPel. SingleDBWD solved only 49% of the problems. These performance differences are statistically significant.

Table 2: Iterations vs. Violations:Graph Colouring Problems.

	StochDisPel		DynAPP		SingleDBWD	
N. Itrs	M viols	% sol	M viols	% sol	M viols	% sol
10	30	0	28	0	84	0
50	14	1	12	**3**	33	0
100	10	2	8	10	21	0
500	4	32	4	57	8	13
1000	4	50	2	70	6	32
1500	3	57	1	69	3	42
2000	2	**68**	1	**93**	6	**49**

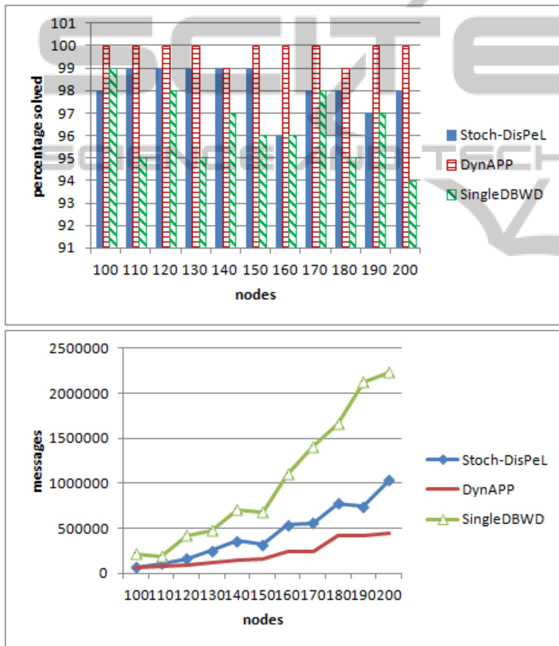

Figure 3: Graph Colouring Problems.

Meeting Scheduling. We also conducted experiments with meeting scheduling problems. A distance chart between locations is randomly generated so that the distance between two locations is assigned a value between 0 and the maximum possible distance indicating the travelling time required. We present results for $60 \leq meetings \leq 140$, with a maximum possible distance (md) of 3 and constraint density (d) between 0.1 and 0.25. The results, presented in Figure 4, are median messages and percentage of problems solved over 100 runs, and show DynAPP's performance was better than that of the other two algorithms. These performance differences are statistically significant.

When comparing the number of iterations versus

Table 3: Iterations Vs Violations:Meeting Scheduling Problems.

	StochDisPeL		DynAPP		SingleDBWD	
N. Itrs	M viols	% sol	M viols	% sol	M viols	% sol
10	3	6	3	**12**	45	0
50	2	55	1	72	3	12
100	1	84	1	86	1	40
500	1	**91**	0	**100**	1	**75**
1000	1	98	*	*	1	82
1400	0	100	*	*	1	87

constraint violations, Table 3 shows that at the first 10 iterations, 12% of the problems were already solved by DynAPP while StochDisPeL solved 6% and SingleDBWD didnot solve any problem. At 500 iterations, DynAPP had solved all the problems, StochDisPeL was able to solve all after 1400 iterations and SingleDBWD solved only 87% of the problems. These performance differences are statistically significant.

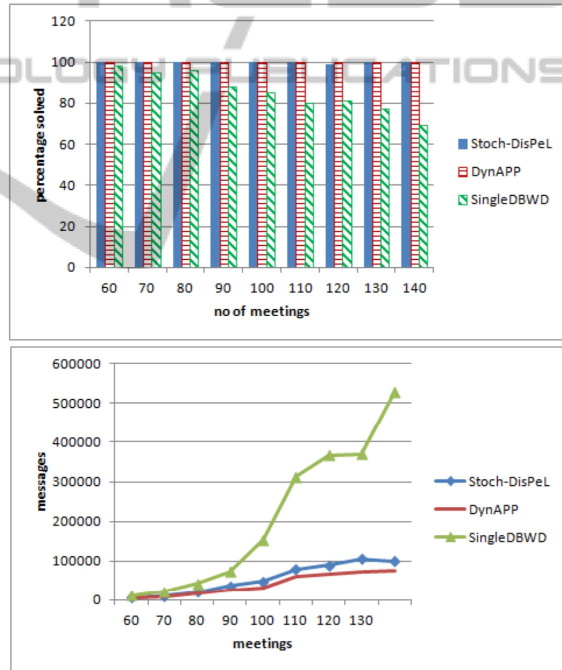

Figure 4: Meeting Scheduling Problems.

5 SUMMARY AND CONCLUSIONS

In this paper, we have presented Dynamic Agent Prioritisation and Penalties (DynAPP), a distributed local search algorithm for solving DisCSP that combines two existing heuristics (penalties on values and agent priority) for escaping local optima. DynAPP significantly improves the performance of distributed

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

local search. *Penalties on values* is a fine-grained heuristic that detects a set of nogood values often found at local optima while *prioritisation* changes the importance of agents.

Empirical results show that DynAPP used less messages and solved more problems on a wide range of random, graph colouring and meeting scheduling problems. We also evaluated DynAPP, SingleDBWD and StochDisPeL to determine the algorithm that optimizes the number of constraints violated given restricted resources (maximum number of iterations). DynAPP was found to perform best by solving the most number of problems at given iteration intervals. We intend to extend our cooperative approach for solving DisCSP with complex local problems, i.e. where agents are responsible for more than one variable.

REFERENCES

Basharu, M., Arana, I., and Ahriz, H. (2006). Stochdispel: exploiting randomisation in dispel. In *Proceedings of 7th International Workshop on Distributed Constraint Reasoning, DCR2006*, pages 117–131.

Basharu, M., Arana, I., and Ahriz, H. (2007a). Escaping local optima: constraint weights vs. value penalties. In *27th SGAI International Conference on Artificial Intelligence, AI-07*, pages 51–64. Springer.

Basharu, M., Arana, I., and Ahriz, H. (2007b). Solving coarse-grained discsps with multi-dispel and disbowd. *Intelligent Agent Technology, IEEE / WIC / ACM International Conference on*, 0:335–341.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Eisenberg, C. (2003). *Distributed Constraint Satisfaction For Coordinating And Integrating A Large-Scale, Heterogeneous Enterprise*. Phd. thesis no. 2817, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland).

Hirayama, K. and Yokoo, M. (2002). Local search for distributed sat with complex local problems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, AAMAS '02, pages 1199–1206, New York, NY, USA. ACM.

Lee, D. A. J. (2010). *Hybrid algorithms for distributed constraint satisfaction.* PhD thesis, School of Computing, The Robert Gordon University, Aberdeen.

Rossi, F., Beek, P. v., and Walsh, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA.

Smith, M. and Mailler, R. (2010). Improving the efficiency of the distributed stochastic algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 1417–1418, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Yokoo, M. (1995). Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 88–102, London, UK. Springer-Verlag.

Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. on Knowl. and Data Eng.*, 10:673–685.

Yokoo, M. and Hirayama, K. (1996). Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of ICMAS-96*.

Zhang, W., Wang, G., and Wittenburg, L. . (2002). Distributed stochastic search for constraint satisfaction and optimization: Parallelism, phase transitions and performance. In *Proceedings of AAAI Workshop on Probabilistic Approaches in Search*.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS