Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

REAL-TIME DECISION MAKING USING SIMULATION

Mukesh Dalal
Brett Groel
Armand Prieditis

LookAhead Decisions Inc.
429 F Street, Suite 4
Davis, CA 95616, U.S.A.

ABSTRACT

Based on a discrete-event simulation model, Simulation-based Real-time Decision-Making (SRDM) is an innovative approach to real-time, goal-directed decision-making. When applied to a flexible manufacturing system, SRDM makes better decisions than most fixed policies, such as deterministic, stochastic and manual. SRDM even improved over fixed policies optimized within a class of policies by OptQuest, in our numerical experiments. Compared to these fixed policies, SRDM shows greater improvement for more complex systems and is quite robust with respect to modeling errors. SRDM provides an improvement over fixed policies by its ability to implement adaptive policies. Since most real-time decisions in currently deployed manufacturing systems are made either manually or by using fixed policies, our results suggest that using SRDM instead could lead to significant improvement in operating performance.

1 INTRODUCTION

Many real-world systems require making *decisions*, but how should *good* decisions be made? Intuitively, good decisions optimize one or more *key performance indicators* (KPIs) such as cost, throughput, lead time, or profit. For example, in a flexible manufacturing plant (Gershwin 1994), a decision may require choosing among alternative routes. In a specific situation, routing the next part to a particular machine might lower the average plant lead time.

Real-time decision-making continuously involves making a decision within the available time. Such decision-making trades off *time* against *quality*: generally, the more deliberation time, the higher the decision quality. However, too much deliberation time can have drastic consequences. For example, inventory might spoil if too much time is spent deliberating.

In a *complex system*, it is usually difficult to calculate the effect of a particular decision on the overall system KPIs. This is because a complex system is likely to involve stochastic processes (machines might have probabilistic run times), complex dependencies among components (one machine might feed to multiple other machines), and uncertain external environment (actual demand may vary over time and may differ from the forecasts). Decision-making in a complex system becomes even more difficult under real-time constraints.

As a result, a *policy* is used instead to decide what to do in a particular situation. For example, a *deterministic* policy might route the part to the *least busy* machine. The goal of the policy is to approximate the decisions that optimize the KPIs. More generally, a policy is a probability distribution function over all the alternative decisions, conditioned by a specific situation. For example, the situation for the part routing decision might be defined in terms of the length of the queues in front of each machine, and the *stochastic* policy might define the probability of choosing a route using a linear combination of the queue lengths. Different coefficients (called *decision variables*) in this linear combination will result in different decision-making behavior, which in turn will affect the resulting KPIs. Thus, creating a policy requires determining the parametric format of the policy (for example, a linear combination of the queue lengths) and then finding the optimal values of the decision variables (for example, the coefficients in this linear combination).

The parametric format of the policy is usually specified by some system expert. The typical approach is to determine the most promising formats based on the expert understanding of the system and then evaluate each of them to determine the best format.

Sometimes, the optimal values of the decision variables are also *manually* specified by the expert. A better approach is to use automated software tools, such as OptQuest (Glover, Kelly et al. 1996), which is a popular nonlinear *optimization* package based on scatter search, tabu search and neural networks. Since many complex systems are difficult to model using analytical techniques like

mathematical programming (Luenberger 1984), these optimization packages use a discrete event simulation system for evaluating the performance. This process of finding the best values of the decision variables based on the output of a simulation model is known as s*imulation optimization* (Law and McComas 2002; Olafsson and Kim 2002). Most commercial simulation systems now offer some simulation optimization capability, for example, SIMUL8, CSIM, and Arena offer OptQuest add-ons.

Apart from the reliance on the expert to specify the correct parametric format, a major problem with a fixed policy (deterministic or stochastic, manual or optimized) is that it may provide a poor approximation to the optimal decision in a specific real-time situation. For example, the policy that always routes a part to the least busy machine does not consider that the least busy machine is taking unusually longer since noon today because of a new operator. This policy also does not consider that several machines just beyond the first machine are extremely busy because of a large order of some other products. Such myopia and rigidity leads a fixed policy to make poor decisions. This paper presents a novel approach, called *Simulation-based Real-time Decision-Making* (SRDM) that removes this myopia and rigidity to make better real-time decisions.

The rest of the paper is organized as follows. Section 2 describes SRDM. Section 3 describes the design of our simulation study. Section 4 presents preliminary results with SRDM. Sections 5 and 6 compare SRDM with expert-created and OptQuest-generated decision policies, respectively. Section 7 discusses related work. Finally, Section 8 summarizes the conclusions of this paper and describes promising areas of future work.

2 SIMULATION-BASED REAL-TIME DECISION-MAKING

Simulation-based Real-time Decision-Making (SRDM) is a dynamic goal-directed decision making process useful for systems that continuously make decisions in real-time in order to optimize some overall system KPIs.

As Figure 1 shows, SRDM relies on a discrete event simulation model of the underlying application. Though the simulation uses a fixed policy (deterministic or stochastic, manual or optimized), SRDM does not use that policy to make a decision in the current situation. Instead it runs several simulations (called *look-aheads)* for a small number of alternative decisions and then selects the decision that optimizes the KPIs. In short, the look-ahead simulations overcome the myopia and rigidity of the underlying fixed policy by taking into account the longer-term impact of each decision in the current situation. Each look-ahead simulation is used to compute the KPIs by combining the KPIs observed during the look-ahead and the KPIs estimated from the terminal situation in that look-ahead.

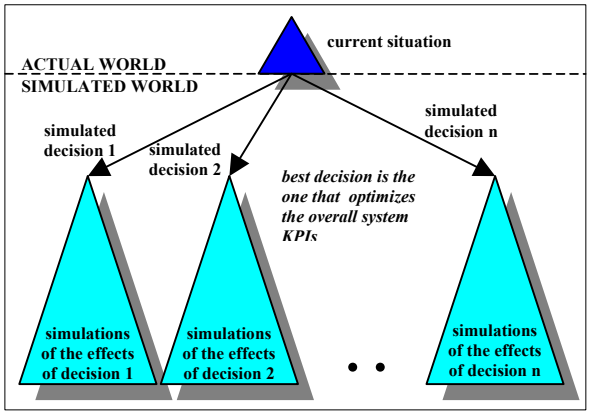

Figure 1: Simulation-Based Real-Time Decision Making

This paper presents a simple variant of SRDM, which is defined by four key parameters:

- *Policy:* Which fixed policy to use during the look-ahead simulations?
- *Depth:* How long to run each look-ahead simulation?
- *Width:* How many look-ahead simulations to run for each decision alternative?
- *Heuristics:* Which heuristics to use to estimate the KPIs at the end of each look-ahead simulation? Heuristics are necessary to estimate the KPIs for the work in progress.

For each decision opportunity, SRDM uses the simulation model to generate the required number of depth-restricted look-ahead simulations for each alternative. The KPIs from these look-aheads are averaged and the decision with the best aggregated KPI is chosen.

The real-time constraint is met as follows: SRDM starts with depth 0, where the fixed policy completely determines the decision. SRDM keeps incrementing the depth until the available time runs out or the depth limit is reached. Finally, it chooses the decision based on the last depth for which all the look-aheads were successfully completed.

More sophisticated versions of SRDM are presented in other papers. For example, one variant interleaves both the depth and width increments to provide decisions with a desired statistical confidence level.

SRDM has been used to develop the *Decision Improver* component of LDI's Rapid Response System (RRS), whose architecture is illustrated in Figure 2. RRS is a general-purpose software used for real-time decision making in wide variety of applications, like manufacturing, healthcare and business processes. By making real-time automated decisions, RRS quickly responds to unexpected real-time events like machine breakdown, patient emergency, traffic congestion, and communication node failure. Since RRS uses a model of the application, it even anticipates and avoids future problems (like traffic congestions) and exploits future opportunities (like product arrivals).

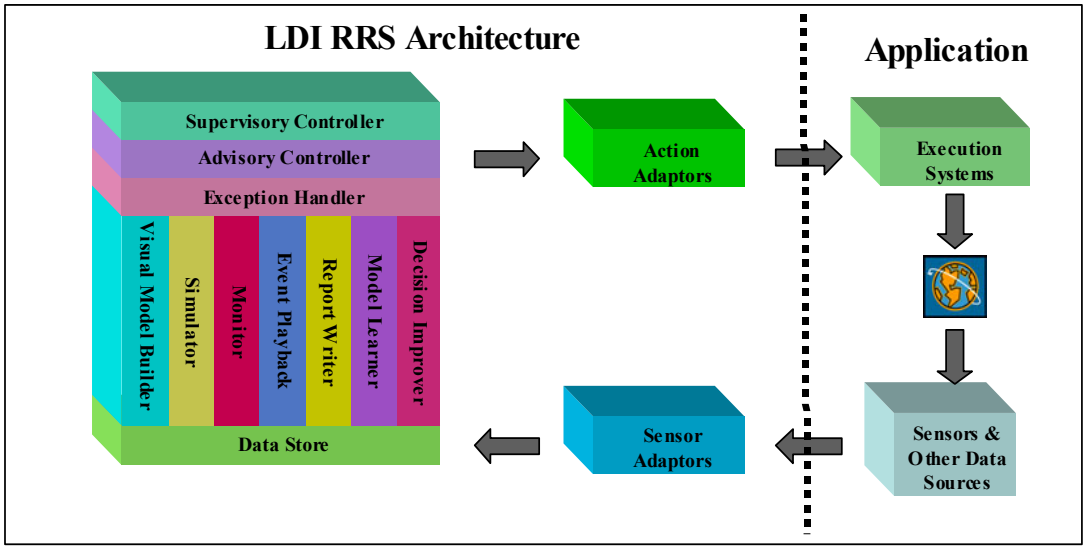

Figure 2: RRS Architecture

Since RRS learns this model over time, it is easy and cost-effective to deploy (no need to manually create this model by expensive domain experts), and it continually improves its performance while automatically adapting to changes. RRS senses and impacts the real world (say, a manufacturing plant) through real-time sensors and execution systems. It learns and uses a simulation models to improve decision making using SRDM. This paper ignores the other components of RRS, for example, it assumes that the simulation model has already been built.

3 SIMULATION STUDY DESIGN

For a simulation study to assess the effectiveness of SRDM, we use the routing problem in a simple reliable flexible (one-part with multiple routings) manufacturing system as shown in Figure 3, which is a variant of a system presented in (Drake and Smith 1996).

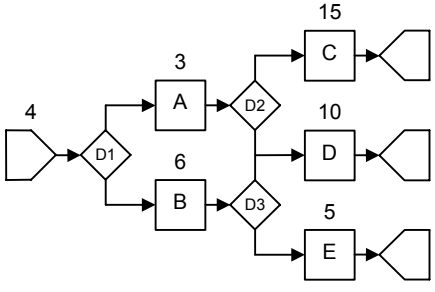

Figure 3: A One-Part, Two-Layer, Multi-Routing Flexible Manufacturing System with Stochastic Arrival and Processing

This system consists of 5 machines (**A** to **E**) arranged in two layers and connected by various route segments. Identical parts arriving from the left side are completely processed when they depart at the right side, after following any of the following alternative routes: **A-C**, **A-D**, **B-D**, or **B-E**. Thus, there are three decision opportunities:

D1. New part: choose either Machine **A** or **B**.
D2. After Machine A: choose either Machine **C** or **D**.
D3. After Machine B: choose either Machine **D** or **E**.

Thus, the decision alternatives are *Left* (A, C, and D, respectively, for the three decisions) or *Right* (B, D, and E, respectively). The route segments as well as the queues in front of each machine are FIFO (first-in-first-out). The operational objective (KPI) is to minimize the *average lead time*, that is, the average time a part spends in the system (from arrival to departure).

In our simulation model, the arrival and processing times are exponentially distributed – Figure 3 also shows the corresponding means (in Minutes). The travel time between each pair of nodes is fixed to 2 Minutes.

There were several reasons for choosing this model, including:

- It represents a very common decision problem in realistic manufacturing systems.
- Its simplicity allows us to clearly understand the effects of various strategies. The simplicity also provided several experimental conveniences regarding debugging, simulation duration, etc.
- It can be made increasingly more complex in a systematic manner, say, by adding more layers of machines. We will exploit this feature to evaluate the scaling of SRDM.
- The complexity of such models does not allow analytic solutions of optimal policies (Gershwin 1994).

Except in one set of experiments described later, we used the same simulation model for look-aheads, OptQuest optimization and experimental comparison of the results. In an actual deployment of RRS, while a manufacturing execution system would replace the last use of simulation, the first two uses will still require building a simulation model.

In each comparison experiment, we conducted enough trials to get at least 99% confidence level that one approach is better than the others (except in one case described later) – this resulted in at least 30 trials in each case. In each trial, the execution system was simulated for at least 1,000 Minutes of simulated time, a long enough time for stable results.

For fairness in comparison, we used identical sequences of random numbers for each pair of simulation trials. These numbers were generated independently from the random numbers used during look-ahead. Further, each machine and the arrivals were based on independent random number sequences.

We used SIMUL8 Standard 9.0 and OptQuest 7.0 for SIMUL8 to get the optimal OptQuest policies, and SLX 0.99 for SRDM and for comparison experiments. While SIMUL8 provides an OptQuest add-on, SLX (Henriksen 2000) provides built-in support for look-ahead.

4 PRELIMINARY SRDM RESULTS

As we described earlier, there are four important SRDM parameters: depth, width heuristics, and policy. In this section, we study the effect of the first three on the performance; the effect of the last one will be studied in the next few sections.

4.1 Effects of Depth and Width

Figure 4 shows the effect of width and depth on the average lead time. As expected, the performance generally improves with increased width and depth. While depths of 8 Minutes or less of simulated time for look-ahead produces poor performance, the performance with greater depths is mostly independent of widths that are 32 or more.

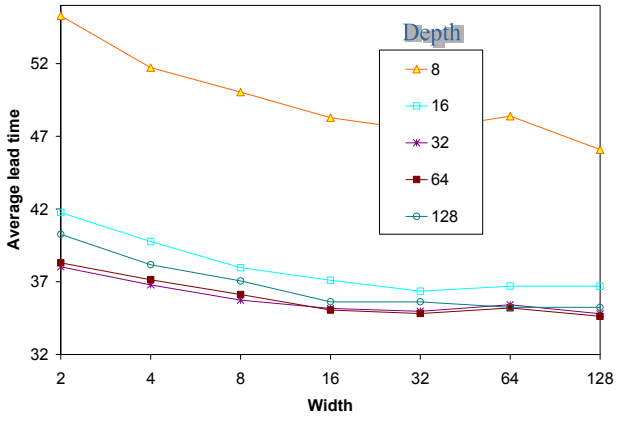

Figure 4: Effects of Depth and Width on Average Lead Time

These results suggest that there is a limit to the depth and width for any system, beyond which the performance does not improve (diminishing returns effect: major increase in depth results in none or minor decrease in average lead time).

Unless explicitly mentioned, we fixed the depth to 64 and width to 32 for the rest of the experiments with this system.

4.2 Effects of Heuristics

Table 1 shows the effects of heuristics (using depth 25 and width 10), which are used at the terminal state of a look-ahead. The *simple* heuristic computes the average time to completely process each remaining part, assuming that there are no other parts in the system. The *complex* heuristic also approximates the delays for each part due to the existing queues in the system. The *none* column shows the results without any heuristics. Results show that these heuristics did not improve the performance. It seems that the approximations in the heuristics were too gross and/or that the look-ahead depth of 25 was sufficient enough to completely evaluate the impact of a decision, without resorting to heuristics.

Table 1: Effects of Heuristics on Average Lead Time

Heuristics	Av. Lead Time
None	35.58
Simple	73.93
Complex	36.92

Although it is still open whether some other heuristics may improve this performance, we did not use any heuristics for any other experiment reported in this paper.

5 COMPARISON WITH MANUAL POLICIES

In this section, we compare the performance of SRDM with two manually-created fixed policies for our experimental model:

- *Deterministic Policy:* Choose the machine with the shortest queue (break ties by choosing the machine on the Right).
- *Stochastic Policy:* The probability of choosing a machine is inversely proportional to its queue length.

The results (Table 2) show that SRDM lowered the average lead time with either policy, with much greater improvement in the stochastic case. SRDM also lowered the variance in both cases, suggesting that it is more robust with respect to the uncertainty in the manufacturing environment (specifically, arrival rates and processing times). Interestingly, both the fixed policies led to similar performance of SDRM, suggesting that SDRM's performance is somewhat independent of the underlying policy.

Table 2: Comparison of Fixed Manual Policies and SRDM in Two Settings – Deterministic and Stochastic

Measure	Deterministic	Stochastic
Avg. Lead Time – Fixed	39.47	47.62
– SRDM	35.56	35.54
– Improvement	**10%**	**25%**
L.T. Variance – Fixed	36.19	93.66
– SRDM	31.79	32.79
Parts Processed –Fixed	241	236
– SRDM	242	242

6 COMPARISON WITH FIXED POLICIES ENHANCED BY OPTQUEST

6.1 OptQuest Optimization

Recall that OptQuest requires the user to provide a parametric format of the policy – it returns the optimal values of the decision variables in that format. Since selecting the right parametric format is still an art, we considered several different formats in order to select the best ones to compare with SRDM. For each OptQuest optimization using this model, we used 30 trials per simulation (1,000 simulated minutes each) and simulations until convergence was visually observed (at least 1,000 simulations). We used the following fixed policy formats:

- *Deterministic local linear:* At D1, choose left (i.e., A) if the expression "xQ(A) + yQ(B) + z" is greater than 50, where Q(M) is the length of the queue for a machine M and x,y,z are the OptQuest decision variables. Similar for D2 and D3.
- *Stochastic local linear:* Use the same expression as above, but divide it by 100 to get the probability of choosing Left (negative values were replaced by 0 and values larger than 1 were replaced by 1).
- *Deep linear (deterministic or stochastic):* Similar to local linear, except that the lengths of all downstream queues are considered to make a decision.
- *Local normalized (deterministic or stochastic):* Similar to local linear , except that the expression for D1 is: "x(Q(A)+1)/(Q(A)+Q(B)+2) + y". Similar for expressions for D2 and D3.

Table 3 shows the average lead time using the optimal policy produced by OptQuest for each of the above fixed policy classes. The results show that deterministic policies outperformed stochastic ones, local linear outperformed deep linear, and local normalized outperformed local linear. It is still open whether some other parametric form will lead to better OptQuest performance.

Table 3: Performance of OptQuest-Optimized Policies

Average lead time	Deterministic	Stochastic
Local linear	40.09	43.87
Local normalized	39.51	40.94
Deep linear	41.04	54.93

6.2 Comparing Fixed and Adaptive Policies

Tables 4 and 5 compares the performance of the OptQuest-optimized policies with SRDM, for local linear and local normalized formats. As before, we used a depth of 64 and a width of 32. The results show that SRDM outperformed OptQuest-optimized policies in all cases, with more improvement with the stochastic and linear policies, as compared to deterministic and normalized policies, respectively. The results also provide more evidence to our earlier observation that SDRM's performance is somewhat independent of the underlying policy. However, OptQuest turned out to be slightly more robust (lower variance) with respect to external uncertainty, except in the stochastic local linear case.

Table 4: Comparing SRDM and OptQuest-Optimized Policies for Local Linear Formats

Measure	Deterministic	Stochastic
Avg. Lead Time – Fixed	40.09	43.87
– SRDM	36.26	34.69
– Improvement	**10%**	**21%**
L.T. Variance – Fixed	30.06	111.63
– SRDM	36.15	34.49
No. parts processed - Fixed	240	237
– SRDM	242	242

Table 5: Comparing SRDM and OptQuest-Optimized Policies for Local Normalized Formats

Measure	Deterministic	Stochastic
Avg. Lead Time – Fixed	39.51	40.94
– SRDM	35.72	34.73
– Improvement	**10%**	**15%**
L.T. Variance – Fixed	30.94	30.40
– SRDM	34.44	34.96
Parts Processed -Fixed	242	240
– SRDM	242	243

6.3 Scaling with System Complexity

In the next set of experiments, we study the effect of scaling up the model complexity on OptQuest and SRDM. For this, we generated three other models, with varying complexity, as shown in Figure 5. Both OptQuest and SRDM were run with parameters that provided them enough flexibility to come up with the best possible results. For example, the number of OptQuest trials was increased to 50 and the SRDM depth was also increased to 100.

Figure 6 plots the performance improvement of adaptive policies implemented by SRDM over fixed policies based on the stochastic local linear policy enhanced by OptQuest, across the four models. It shows that SRDM provides more improvement in the more complex models

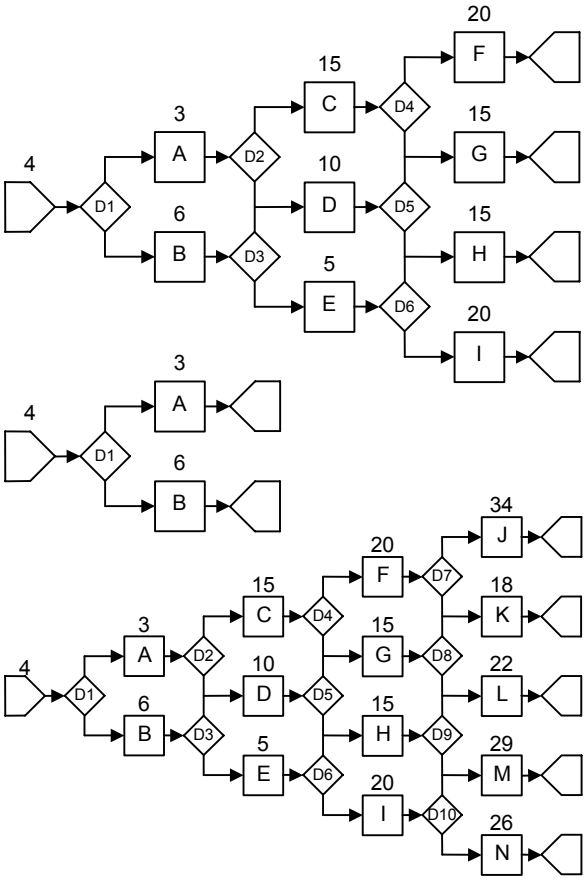

Figure 5: Three, One, and Four-Layer One-Part Flexible Manufacturing Systems with Multiple Routings

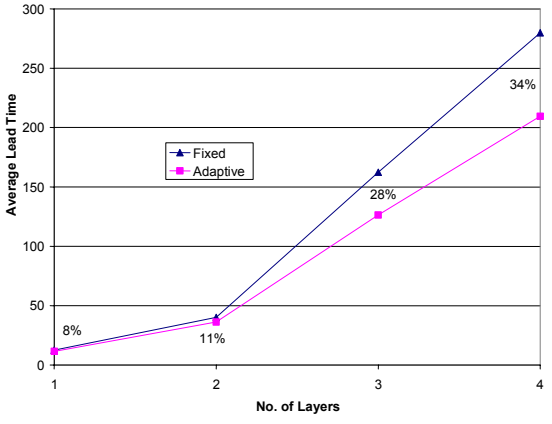

Figure 6: Performance Scaling of Fixed and Adaptive Policies

tested in this experiment. This suggests that SRDM might provide significant improvements in real-world systems, since they are typically more complex than the models considered here. Note that as problem complexity increases, deeper and wider look-aheads might be required. Indeed, our results showed that the time per decision also increases with model complexity.

6.4 Processing Time

The processing time taken by OptQuest and SRDM directly depends on the specifications of the host machine. Our experiments used a 2.25 GHz single-processor machine with 512 MB of RAM. To analyze the processing time, we examined the average CPU time per decision. The OptQuest average CPU time per decision is essentially zero, because each decision using an OptQuest generated policy requires at most one linear function evaluation. However, OptQuest requires several hours of offline processing time to determine the optimal coefficients for the policy function. SRDM requires no offline processing time, and Table 6 summarizes the SRDM average CPU time per decision for each model.

Table 6: SRDM Average CPU Time Per Decision

Model Layers	Average CPU Time Per Decision
1	0.015 s
2	0.026 s
3	0.041 s
4	0.061 s

The results show that the average CPU time per decision increases with the model complexity. However, even for the 4-layer model, the average CPU time per decision is still less than 1/10 of a second, which is small enough to be viable for a real-time decision-making system. Processing time can be further reduced by distributing the calculations across multiple machines or by dividing a large model into smaller sub-models.

7 OTHER RESULTS

By varying some experimental parameters, we did some ad-hoc sensitivity analysis of our results, using the original two-layer system and deterministic local linear policy:

- *Effect of processing and arrival times:* We generated two more sets of these stochastic distributions. For a congested system with means (3, 5, 6, 9, 8, 10), the average lead times for OptQuest and SRDM were 78.10 and 74.57, respectively – a 5% improvement (with 95% confidence). For a less congested system with means (5, 5, 6, 10, 4, 9), the average lead times for OptQuest and SRDM were 29.70 and 25.27, respectively – a 15% improvement.
- *Effect of erroneous distributions:* We changed the mean processing times for the simulation of the execution system, but not for the look-ahead and OptQuest optimization (thus, both look-ahead and OptQuest used the same erroneous distribution). For two such systems (System 1 and System 2), the errors and the results are shown in Table 7. The results suggest that SRDM is more stable in presence of errors.

Table 7: An Adaptive Policy Outperforms a Fixed Policy Even When Both Use Erroneous Models

Parameter	**System 1**	**Model**	**System 2**
Arrival rate	4	4	4
Machine A	1.5	3	4.5
Machine B	9	6	3
Machine C	7.5	15	22.5
Machine D	15	10	5
Machine E	2.5	5	7.5
OptQuest	44.00	Used by SRDM and OptQuest	136.60
SRDM	35.54		28.49
Improvement	**19%**		**79%**

- *Effect of longer Simulation runs:* Instead of using 1,000 simulated Minutes to run both OptQuest as well as execution system simulations, we tried the following variants (the look-ahead depth remains 64 in all cases):
  1. *The execution system simulations were run for 2,000 Minutes:* The average lead times for OptQuest and SRDM were 43.02 and 37.11, respectively – a 14% improvement.
  2. Both *the execution system and OptQuest simulations were run for 2,000 Minutes:* The average lead times for OptQuest and SRDM were 41.78 and 37.22, respectively – an 11% improvement.

  The results show that the adaptive policies of SRDM continue to outperform the best policy identified by the OptQuest tool from a class of fixed policies that we selected for these experiments.

8 RELATED WORK

Simulation optimization algorithms, like OptQuest, determine the optimal parameters of a pre-specified policy using several simulation runs with different parameter values. This is quite different from SRDM's use of simulation in run-time evaluation and comparison of the alternatives. Although these algorithms are mostly used off-line, some research has been done on the online use of those based on infinitesimal perturbation analysis (Glasserman 1991).

Online simulations have also been used in real-time systems for control, planning, scheduling, etc. (Smith, Wysk et al. 1994; Gonzalez and Davis 1997). Emulation also requires online simulations (McGregor 2002).

Zee (2001) recently presented a look-ahead strategy for real-time adaptive solution of the *batching problem*, where the decision at each moment requires answering the question "to start the machine now or to wait for the next customer to arrive". It reported significant improvement in performance, but like the earlier work on look-ahead strategies starting from Glassey and Weng (1991), uses heuristics that require the knowledge of a few near future arrivals. In contrast, SRDM does not need this information. Moreover, this heuristics-based approach is fundamentally different from SRDM's simulation-based approach.

Rather than using as simulation optimization system to find the policy, another approach is to *learn* a policy from example decisions made in the actual environment (Damerdji 1993; Mahadevan, Marchalleck et al. 1997; Mahadevan and Theocharous 1998; Schneider, Boyan et al. 1998; Riedmiller and Riedmiller 1999; Miyashita 2000). This method, known as *reinforcement learning*, uses feedback from the environment (simulated or real) to update the policy after each decision. For example, Russell and Norvig (1995) survey several reinforcement learning techniques, which learn a deterministic policy through single-depth simulation in the environment. Another technique, known as *policy iteration,* picks a deterministic policy and then calculates the performance indicator for each situation given the policy. It repeatedly chooses the best policy for each situation based on the previous policy assignment until the policy stabilizes. For either reinforcement learning or policy iteration, the resulting policy will suffer from the same problems of rigidity and myopia as a policy found through simulation optimization. In general, the source of the fixed policy does not eliminate the problems inherently associated with its use.

9 CONCLUSIONS AND FUTURE WORK

We described a new approach (SRDM) for real-time goal-directed decision-making and compared it with fixed-policies using a flexible manufacturing simulation study. Our results show that SRDM significantly improved over several fixed policies: deterministic, stochastic, local, deep, linear, normalized, manually-created, and even optimized with OptQuest within the classes of policies considered here. While coming up with the best parametric form for optimization is quite challenging, especially for complex applications, this is not a problem for SRDM, since its performance was almost independent of the underlying policy.

Our results also show that SRDM provides more improvement for more complex systems, though it takes longer to make each decision. We see this increase in decision time as the biggest potential problem in using SRDM for very complex applications. Since the focus of this paper was on KPI improvement, we have not tuned our SRDM implementation to optimize the decision time. Moreover, the low decision times observed in our experiments, even for the relatively complex 4-layer network, suggests that the current implementation of SRDM might be practical for many real-world applications.

Our results also show that SRDM is quite robust with respect to modeling inaccuracies. This is an important practical consideration, since the models do not accurately reflect the reality. Our future work includes learning and tuning the model based on observation from the real system.

ACKNOWLEDGMENTS

This work was performed under the support of the U.S. Department of Commerce, National Institute of Standards and Technology, Advanced Technology Program, Cooperative Agreement Number 70NANB1H3034. We thank Wolverine Software Corp. and SIMUL8 Corp. for providing their software and technical support, and OptTek Sytems Inc. for quickly fixing a bug we discovered in OptQuest. We are also grateful to Wolverine Software Corp. for modifying their SLX system to support look-ahead. The errors and omissions in this paper are the sole responsibility of the authors.

REFERENCES

Damerdji, H. 1993. Parametric Inference for Generalized Semi-Markov Processes. *Proceedings of the Winter Simulation Conference*, G. W. Evans, M. Mollaghasemi, E. C. Russell, and W. E. Biles, eds., Los Angeles, California, USA, 323-328.

Drake, G. and J. S. Smith. 1996. Simulation System for Real-Time Planning, Scheduling, and Control. *Proceedings of the 1996 Winter Simulation Conference*.

Gershwin, S. B. 1994. *Manufacturing Systems Engineering*. Prentice Hall.

Glasserman, P. 1991. *Gradient Estimation via Perturbation Analysis*. Kluwer Academic Publishers.

Glassey, C. R. and W. W. Weng. 1991. Dynamic batching heuristic for simultaneous processing. *IEEE Transactions on Semiconductor Manufacturing*, **4,** 77-82.

Glover, F., J. P. Kelly, and M. Laguna. 1996. New Advances and Applications of Combining Simulation and Optimization. *Proceedings of the 1996 Winter Simulation Conference*, J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, eds., 144-152.

Gonzalez, F. G. and W. J. Davis. 1997. A simulation-based controller for distributed discrete-event systems with application to flexible manufacturing. *Proceedings of the 1997 Winter Simulation Conference*, S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, eds., 845-853.

Henriksen, J. O. 2000. SLX: The X is for Extensibility. *Proceedings of the 2000 Winter Simulation Conference*, J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds., 183-190.

Law, A. M. and M. G. McComas. 2002. Simulation-Based Optimization. *Proceedings of the 2002 Winter Simulation Conference*, E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds., 41-44.

Luenberger, D. 1984. *Introduction to Linear and Nonlinear Programming*. Addison Wesley.

Mahadevan, S. and G. Theocharous. 1998. Optimizing Production Manufacturing using Reinforcement Learning. *Proceedings of the 11th International FLAIRS Conference*, AAAI Press, 372-377.

Mahadevan, S., N. Marchalleck, T. K. Das, and A. Gosavi. 1997. Self_Improving Factory Simulation Using Continuous-Time Average-Reward Reinforcement Learning. *Proceedings of the International Confernce on Machine Learning*, Nashville, TN, Morgan Kaufmann.

McGregor, I. 2002. The relationship between simulation and emulation. *Proceedings of the 2002 Winter Simulation Conference*, E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds., 1683-1688.

Miyashita, K. 2000. Job-Shop Scheduling with Genetic Programming. *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, 505-512.

Olafsson, S. and J. Kim. 2002. Simulation Optimization. *Proceedings of the 2002 Winter Simulation Conference*, E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds., 79-84.

Riedmiller, S. and M. Riedmiller. 1999. A Neural Reinforcement Learning Approach to Learn Local Dispatching Policies in Production Scheduling. *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence*, Stockholm, Sweden, Morgan Kaufmann, 764-769.

Russell, S. and P. Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

Schneider, J., J. Boyan, and A. Moore. 1998. Value Function Based Production Scheduling. *Proceedings of the International Conference on Machine Learning*, Morgan Kaufmann.

Smith, J. S., R. A. Wysk, D. T. Sturrock, S. E. Ramaswamy, G. D. Smith, and S. B. Joshi. 1994. Discrete Event Simulation for Shop Floor Control. *Proceedings of the Winter Simulation Conference*, Lake Buena Vista, FL, 962-969.

Zee, D.-J. v. d. 2001. Real-time adaptive control of multi-product multi-server bulk service processes. *Proceedings of the 2001 Winter Simulation Conference*, B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds., IEEE Computer Society, 930-936.

AUTHOR BIOGRAPHIES

MUKESH DALAL is a Senior Scientist at LookAhead Decisions Inc. He has a B.Tech. degree from Indian Institute of Technolgy, and a M.S. and Ph.D from Rutgers University in New Jersey, all in Computer Science. Before LDI, he was an Assistant Professor at Columbia University in New York, and then a Senior Member at i2 Technologies in Dallas. His research interests include decision making, optimization, artificial intelligence, real-time systems, and supply chain management. His email address is: <Mukesh@LookAheadDecisions.Com>.

BRETT GROEL is a Scientist at LookAhead Decisions Inc. He has a B.S. degree in Computer Science from the University of California at Davis. His undergraduate research projects involved distributed computing. His current interests include simulation, graphics, and visualization. His e-mail address is: <Brett@LookAheadDecisions.Com>.

ARMAND PRIEDITIS is a Senior Scientist at LookAhead Decisions Inc. He has a B.S. degree from the University of Minnesota, Minneapolis, and a M.S. and Ph.D from Rutgers University in New Jersey, all in Computer Science. Before LDI, he was an Assistant Professor at the University of California-Davis and CEO of Unconventional Wisdom. His research interests include decision making, search, heuristics, and machine learning. His email address is: <Prieditis@LookAheadDecisions.Com>.