Systematic Integration of Parameterized Local Search into Evolutionary Algorithms

Neal K. Bambha, Shuvra S. Bhattacharyya
ECE Department and UMIACS
University of Maryland, College Park
{nbambha, ssb}@eng.umd.edu

Jürgen Teich
Computer Engineering
University of Paderborn
Paderborn, Germany
teich@date.uni-paderborn.de

Eckart Zitzler
Computer Engineering
Swiss Federal Institute of Technology
Zurich, Switzerland
zitzler@tik.ee.ethz.ch

Abstract

Application-specific, parameterized local search algorithms (PLSAs), in which optimization accuracy can be traded off with run-time, arise naturally in many optimization contexts. We introduce a novel approach, called simulated heating, for systematically integrating parameterized local search into evolutionary algorithms (EAs). Using the framework of simulated heating, we investigate both static and dynamic strategies for systematically managing the trade-off between PLSA accuracy and optimization effort. Our goal is to achieve maximum solution quality within a fixed optimization time budget. We show that the simulated heating technique better utilizes the given optimization time resources than standard hybrid methods that employ fixed parameters, and that the technique is less sensitive to these parameter settings. We apply this framework to three different optimization problems, compare our results to the standard hybrid methods, and show quantitatively that careful management of this trade-off is necessary to achieve the full potential of an EA/PLSA combination.

1 Introduction

For many optimization problems, efficient algorithms exist for refining arbitrary points in the search space into better solutions. Such algorithms are called *local search algorithms* because they define neighborhoods, typically based on initial “coarse” solutions, in which to search for optima. Many of these algorithms are parameterizable in nature. Based on the values of one or more algorithm parameters, such a *parameterized local search algorithm (PLSA)* can trade off time or space complexity for optimization accuracy.

PLSAs and evolutionary algorithms (EAs) have complementary advantages. EAs are applicable to a wide range of problems, they are robust, and are designed to sample a large search space without getting stuck at local optima. Problem-specific PLSAs are often able to converge rapidly towards local minima. The term ‘local search’ generally applies to methods that cannot escape these minima. For these reasons, PLSAs can be incorporated into EAs in order to increase the efficiency of the optimization.

Several techniques for incorporating local search have been reported. These include Genetic Local Search [28], Genetic Hybrids [14], Random Multi-Start [21], GRASP [12], and others. These techniques are often demonstrated on well-known problem instances where either optimal or near-optimal solutions are known. The optimization goal of these techniques is then to obtain a solution very close to the optimum with acceptable run-time. In this regard, the incorporation of local search has been quite successful. For example, Vasquez and Whitley [36] demonstrated results within 0.75% of the best-known results for the Quadratic Assignment Problem using a hybrid approach, with all run-times under five hours. In most of these hybrid techniques the local search is run with fixed parameter values (i.e. at the highest accuracy setting). In this paper, we consider a different optimization goal, which has not been addressed so far. Here we are interested in generating a solution of maximum quality within a specified optimization time, where the optimization run-time is an important constraint that must be obeyed. Such a fixed optimization time budget is a realistic assumption in practical optimization scenarios. Many such scenarios arise in the design of embedded systems. Later in the paper we give examples of problems for optimizing memory and power in embedded systems. In a typical design process, the designer begins with only a rough idea of the system architecture, and first needs to assess the effects of a large number of design choices—different component parts, memory sizes, different software implementations, etc. Since the time to market is very critical in the embedded system business, the design process is on a strict schedule. In the first phases of the design process, it is essential to get good estimates quickly so that these initial choices can be made. Later, as the design process converges on a specific hardware/software solution, it is important to get more accurate solutions. In these cases, the designer really needs to have the run-time as an input to the optimization algorithm.

In order to accomplish this goal, we vary the parameters of the local search during the optimization process in order to trade off accuracy for reduced complexity. Our optimization approach is general enough to hold for any kind of global search algorithm; however, in this paper we test hybrid solutions that solely use an EA as the global search algorithm. Existing hybrid techniques fix the local search at a single point, typically at the highest accuracy. In the following discussion and experiments, we refer to this method as a *fixed parameter method*. We will compare our results against this method.

One of the central issues we examine is how the computation time for the PLSA should be allocated during the course of the optimization. More time allotted to each PLSA invocation implies more thorough local optimization at the expense of a smaller number of achievable function evaluations (e.g., smaller numbers of generations explored with evolutionary methods), and vice-versa. Arbitrary management of this trade-off between accuracy and run-time of the PLSA is not likely to generate optimal results. Furthermore, the proportion of time that should be allocated to each call of the local search procedure is likely to be highly problem-specific and even instance-specific. Thus, dynamic adaptive approaches may be more desirable than static approaches.

In this paper, we describe a technique called *simulated heating* [37], which systematically incorporates parameterized local search into the framework of global search. The idea is to increase the time allotted to each PLSA invocation during the optimization process — low accuracy of the PLSA at the beginning and high accuracy at the end[1]. This is in contrast to most existing hybrid techniques, which consider a fixed local search function, usually operating at the highest accuracy. Within the context of simulated heating optimization, we consider both static and dynamic strategies for systematically increasing the PLSA accuracy and the corresponding optimization effort. Our goals are to show that careful management of this trade-off is necessary to achieve the full potential of a EA/PLSA combination, and to develop an efficient strategy for achieving this trade-off management. We show that, in the context of a fixed optimization time budget, the simulated heating technique performs better than using a fixed local search.

In most heuristic optimization techniques, there are some parameters that must be set by the user. In many cases, there are no clear guidelines on how to set these parameters. Moreover, the optimal parameters are often dependent on the exact problem specification. We show that the simulated heating technique, while still requiring parameters to be set by the user, is less sensitive to the parameter settings.

We demonstrate our techniques on the well-known binary knapsack problem, and on two optimization prob-

1. In contrast to [37], the time budget here refers to the overall GSA/PLSA hybrid, not only the time resources needed by the PLSA.

lems for embedded systems which have quite different structure.

2 Related Work

In the field of evolutionary computation, hybridization seems to be common for real-world applications [16] and many evolutionary algorithm/local search method combinations can be found in the literature, e.g., [11, 18, 33, 27, 38]. Local search techniques can often be incorporated naturally into evolutionary algorithms (*EA*s) in order to increase the effectiveness of optimization. This has the potential to exploit the complementary advantages of EAs (generality, robustness, global search efficiency), and problem-specific PLSAs (exploiting application-specific problem structure, rapid convergence towards local minima).

As we will explain in more detail in section 5, a hybrid evolutionary algorithm/local search method was applied to a memory optimization problem in embedded systems [38]. This method employed the standard, fixed parameter approach. We will parameterize this local search and use it in simulated heating in this paper. Below we list some other hybrid methods, and suggest how they could potentially be adapted to use our simulated heating technique.

One problem to which hybrid approaches have been successfully applied is the quadratic assignment problem (QAP), which is an important combinatorial optimization problem. Several groups have used hybrid genetic algorithms that are effective in solving the QAP. The QAP concerns n facilities, which must be assigned to n locations at minimum cost. The problem is to minimize the quantity $C(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)}$, $\pi \in \Pi(n)$, where $\Pi(n)$ is a set of all permutations of $\{1, 2, \ldots, n\}$, a_{ij} are elements of a distance matrix, and b_{ij} are elements of a flow matrix representing the flow of materials from facility i to facility j.

Merz and Freisleben [28] presented a Genetic Local Search (GLS) technique, which applies a variant of the *2-opt* heuristic as a local search technique. For the QAP, the 2-opt neighborhood is defined as the set of all solutions that can be reached from the current solution by swapping two elements of the permutation π. The size of this neighborhood increases quadratically with n. The 2-opt local search employed by Merz takes the first swap that reduces the total cost $C(\pi)$. This is done to increase efficiency.

Fleurent and Ferland [14] combined a genetic algorithm with a local Tabu Search (TS) method. In contrast to the simpler local search of Merz, the idea of the TS is to consider all possible moves from the current solution to a neighboring solution. Their method is called Genetic Hybrids. They improved the best solutions known at the time for most large scale QAP problems.

By comparison, simulated heating for QAP might be formulated as a combination of the above two methods. One could consider the best of m moves found that reduce $C(\pi)$, where m is the PLSA parameter.

Vazquez and Whitley [36] also presented a technique, which combines a genetic algorithm with TS, where the genetic algorithm is used to explore in parallel several regions of the search space and uses a fixed Tabu local search to improve the search around some selected regions. They demonstrated near optimal performance, within 0.75% of the best known solutions. They did not investigate their technique in the context of a fixed optimization time budget.

Random multi-start local search has been one of the most commonly used techniques for combinatorial optimization problems [21][32]. In this technique, a number of solutions are generated randomly at each step, local search is repeated on these solutions, and the best solution found during the entire optimization is output. Several improvements over random multi-start have been described. Greedy randomized adaptive search procedures (GRASP) combine the power of greedy heuristics, randomization, and conventional local search procedures [12]. Each GRASP iteration consists of two phases—a construction phase and a local search phase. During the construction phase, each element is selected at random from a list of candidates determined by an adaptive greedy algorithm. The size of this list is restricted by parameters α and β, where α is a value restriction and β is a cardinality restrictions. Feo et al. demonstrate the GRASP technique on a single machine scheduling problem [13], a set covering problem, and a maximum independent set problem [12]. They run the GRASP for several fixed values of α and β, and show that the optimal parameter values are problem dependent. In simulated heating, α and β would be candidates for parameter adaptation. In the second phase of GRASP, a local search is applied to the constructed solution to find a local optimum. For the set covering problem, Feo et al. define a k, p exchange local search where all k-tuples in a cover are exchanged with a p-tuple. Here, k was fixed during optimization. In a simulated heating optimization, k might be used as the PLSA parameter, with smaller tuples being exchanged at the beginning of the optimization and larger tuples examined at the end. A similar k-exchange local search procedure was used for the maximum independent set problem.

Kazarlis et al. [20] demonstrate a microgenetic algorithm (MGA) as a generalized hill-climbing operator.

The MGA is a GA with a small population and a short evolution. The main GA performs global search while the MGA explores a neighborhood of the current solution provided by the main GA, looking for better solutions. The main advantage of the MGA is its ability to identify and follow narrow ridges of arbitrary direction leading to the global optimum. Applied to simulated heating, MGA could be used as the local search function with the population size and number of generations used as PLSA parameters.

J. He and J. Xu [17] describe three hybrid genetic algorithms for solving linear and partial differential equations. The hybrid algorithms integrate the classical successive over relaxation (SOR) with evolutionary computation techniques. The recombination operator in the hybrid algorithms mixes two parents, while the mutation operator is equivalent to one iteration of the SOR method. A relaxation parameter ω for the SOR is adapted during the optimization. He and Xu observe that it is very difficult to estimate the optimal ω, and that the SOR is very sensitive to this parameter. Their hybrid algorithm does not require the user to estimate the parameter; rather, it is evolved during the optimization. Different relaxation factors are used for different individuals in a given population. The relaxation factors are adapted based on the fitness of the individuals. By contrast, in simulated heating all members of a given population are assigned the same local search parameter at a given point in the optimization.

When employing PLSAs in the context of many optimization scenarios, however, a critical issue is how to use computational resources most efficiently under a given optimization time budget (e.g., a minute, an hour, a day, etc.). Goldberg and Voessner [16] study this issue in the context of a fixed local search time. They idealize the hybrid as consisting of steps performed by a global solver G, followed by steps by a local solver L, and a search space as consisting of basins of attraction that lead to acceptable targets. Using this, they are able to decompose the problem of hybrid search, and to characterize the optimum local search time that maximizes the probability of achieving a solution of a specified accuracy.

Here, we consider both fixed and variable local search time. The issue of how to best manage computational resources under a fixed time budget translates into a problem of appropriately reconfiguring successive PLSA invocations to achieve appropriate accuracy/run-time trade-offs as optimization progresses.

3 Simulated Heating

From the discussion of prior work we see that one weakness of many existing approaches is their sensitivity to parameter settings. Also, excellent results have been achieved through hybrid global/local optimization techniques, but they have not been examined carefully for a fixed optimization time budget. In the context of a limited time budget, we are especially interested in minimizing wasted time. One obvious place to focus is at the beginning of the optimization, where many of the candidate solutions generated by the global search are of poor quality. Intuitively, one would want to evaluate these initial solutions quickly and not spend too much time on the local search. Also, it is desirable to reduce the number of trial runs required to find an optimal parameter setting. One way to do this is to require only that a good *range* for the parameter be given. These considerations lead to the idea of simulated heating.

3.1 Basic Principles

A general single objective optimization problem can be described as an objective function f that maps a tuple of m parameters (decision variables) to a single objective y. Formally, we wish to either minimize or maximize $y = f(\vec{x})$ subject to $\vec{x} = (x_1, x_2, \dots, x_m) \in X$ where $\vec{x}$ is called the *decision vector*, X is the *parameter space* or *search space*, and y is the objective. A solution candidate consists of a particular $(y_0, \vec{x}_0)$ where $y_0 = f(\vec{x}_0)$.

We will approach the optimization problem by using an *iterative search process*. Given a set X, and a function F, which maps X onto itself, we define an iterative search process as a sequence of successive approximations to F, starting with an x^0 from X, with $x^{r+1} = F(x^r)$ for $r = (0, 1, 2, \dots)$. One *iteration* is defined as a consecutive determination of one candidate from another candidate set using some F. For an evolutionary algorithm, one iteration consists of the determination of one generation from the previous generation, with F consisting of the selection, crossover, and mutation rules.

The basic idea behind simulated heating is to vary the local search parameter p during the optimization process. This is in contrast to the more commonly employed technique of choosing a single value for p (typically that value producing highest accuracy of the local search $L(p)$) and keeping it constant during the entire optimization. Here, we start with a low value for p, which implies a low cost $C(p)$, and accuracy $A(p)$ for the local search, and increase p at certain points in time during the optimization, which increases $C(p)$ and $A(p)$. This is depicted in Figure 1, where the dotted line corresponds to simulated heating, and the dashed line corresponds to the traditional approach. The goal is to focus on the global search at the beginning and to find promising regions of the search space first; for this phase, $L(p)$ runs with low accuracy, which in turn allows a greater number of optimization steps of the

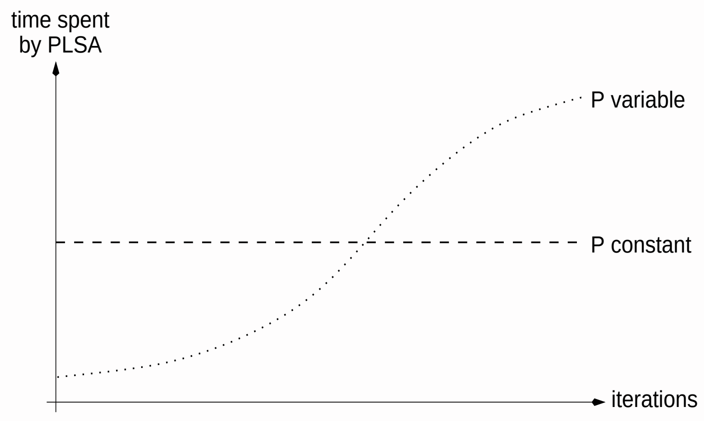


Figure 1. Simulated heating vs. traditional approach to utilizing local search.

global search G. Afterwards, more time is spent by $L(p)$ in order to improve the solutions found or to assess them more accurately. As a consequence, fewer global search operations are possible during this phase of optimization. Since $A(p)$ is systematically increased during the process, we use the term *simulated heating* for this approach by analogy to simulated annealing where the 'temperature' is continuously decreased according to a given cooling scheme.

3.2 Optimization Scenario

We assume that we have a global search algorithm (GSA)[2] G operating on a set of solution candidates and a PLSA $L(p)$, where p is the parameter of the local search procedure[3]. Let

- C_{fix} define the maximum (worst-case) time needed by G to generate a new solution that is inserted in the next solution candidate set,
- $C(p)$ denote the complexity (worst-case run-time) of L for the parameter choice p,
- $A(p)$ be the accuracy (effectiveness) of L with regard to p, and
- R denote the set of permissible values for parameter p. Typically, R may be described by an interval $[p_{\text{min}} \ldots p_{\text{max}}] \cap \Re$ where $\Re$ denotes the set of reals and $C(p_{\text{min}}) \le C(p_{\text{max}})$.

Furthermore, suppose that for any pair (p_1, p_2) of parameter values we have that

$$(p_1 \le p_2) \Rightarrow (C(p_1) \le C(p_2)) \textbf{ and } (A(p_1) \le A(p_2)). \tag{1}$$

2. In this paper, we focus on an evolutionary algorithm as the global search algorithm, although the approach is general enough to hold for any global search algorithm.
3. For simplicity it is assumed here that p is a scalar rather than a vector of parameters.

That is, increasing parameter values in general result in increased consumption of compile-time, as well as increased optimization effectiveness.

Generally, it is very difficult, if not impossible, to analytically determine the functions $C(p)$ and $A(p)$, but these functions are useful conceptual tools in discussing the problem of designing cooperating GSA/PLSA combinations. The techniques that we explore in this paper do not require these functions to be known. The only requirement we make is that the monotonicity property (1) be obeyed at least in an approximate sense (fluctuations about relatively small variations in parameter values are admissible, but significant increases in the PLSA parameter value should correspond to increasing cost and accuracy). Consequently, a tunable trade-off emerges: when $A(p)$ is low, refinement is generally low as well, but not much time is consumed ($C(p)$ is also low). Conversely, higher $A(p)$ requires higher computational cost $C(p)$. We define simulated heating as follows:

Definition 1: [Heating scheme]

A heating scheme H *is a triple* $H = (H_R, H_{it}, H_{set})$ *where*:

- H_R *is a vector of PLSA parameter values with* $H_R = (p_1, \ldots, p_n)$, $p_i \in [p_{min}, \ldots, p_{max}]$, *and* $p_1 \le p_2 \le \ldots \le p_n$,
- H_{it} *is a boolean function, which yields true if the number of iterations performed for parameter* p_i *does not exceed the maximum number of iterations allowed for* p_i*; and*
- H_{set} *is a boolean function, which yields true if the size of the solution candidate set does not exceed the maximum size for* p_i *and iteration* t *of the overall GSA/PLSA hybrid.*

The meanings of the functions H_{it} and H_{set} will become clear in the *G/L* hybrid algorithm of Figure 2, which is taken as the basis for the optimization scenario considered in this paper.

The GSA considered here is an evolutionary algorithm (EA) that is

1. Generational, i.e., at each evolution step an entirely new population is created. This is in contrast to a non-generational or steady-state EA that only considers a single solution candidate per evolution step;
2. Baldwinian, i.e., the solutions improved by the PLSA are not re-inserted in the population. This is in contrast to a Lamarckian EA, in which solutions would be updated after PLSA refinement.

4 Simulated Heating Schemes

We are interested in exploring optimization techniques in which the overall optimization time is fixed and

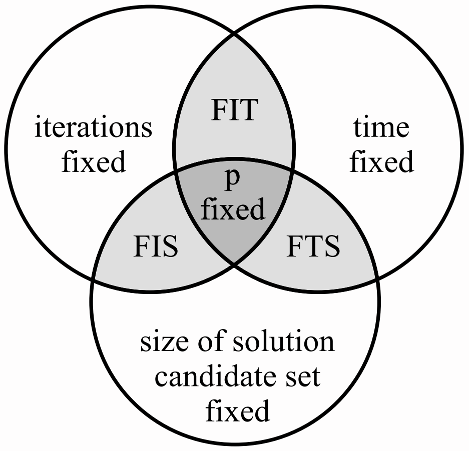
```
Global/Local Hybrid()
Input:    $H = ((p_1, \ldots, p_n), H_{it}, H_{set})$ (heating scheme)
          $T_{max}$                  (maximum time budget)
Output:   s                          (best solution found)

Step 1:   Initialization: Set $T = 0$ (time used), $t = 0$ (iterations performed), and $i = 1$ (current
          PLSA parameter index).
Step 2:   Heating: Set $p = p_i$.
Step 3:   Next iteration: Create an empty multi-set of solution candidates $S_t = \emptyset$.
Step 4:   Global search: If $t = 0$, create a solution candidate $s$ at random. Otherwise, generate a
          new solution candidate using $G$ based on the previous solution candidate set $S_{t-1}$ and the
          associated quality function $F_{t-1}$.
Step 5:   Local search: Apply $L$ with parameter $p$ to $s$ and assign it a quality (fitness) $F_t(s)$.
Step 6:   Termination for candidate set: Set $S_t = S_t + \{s\}$ and $T = T + C_{fix} + C(p)$. If the condi-
          tions $H_{set}$ is fulfilled and $T \le T_{max}$ then go to Step 4.
Step 7:   Termination for iteration: Set $t = t + 1$. If the condition $H_{it}$ is fulfilled and $T \le T_{max}$ then
          go to Step 3.
Step 8:   Termination for algorithm: If $i < n$ increment $i$. If $T \le T_{max}$ then go to Step 2.
Step 9:   Output: Apply $L$ with parameter $p_{max}$ to the best solution in $\bigcup_{1 \le i \le t} S_t$ regarding the corre-
          sponding quality functions $F_i$; the resulting solution $s$ is the outcome of the algorithm.
```

Figure 2. Global/Local Search Hybrid.

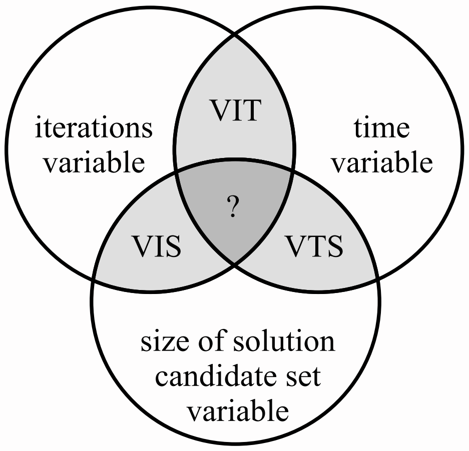
specified in advance (fixed time budget). During the optimization and within this time budget, we allow a heating scheme to adjust three optimization parameters per PLSA parameter value:

1. the number of GSA iterations t_p,
2. the size of the solution candidate set N_i, and
3. the maximum optimization time using this parameter value T_i.

We distinguish between static and dynamic heating based how many of the parameters are fixed and how many are allowed to vary during the optimization. This is illustrated in Figure 3. In our experiments, we keep the size of the solution candidate set (GA population) fixed, and thus only consider the FIS, FTS, and VIT strategies. For the sake of completeness, however, we outline all these strategies below.



i



ii

Figure 3. Illustration of the different types of *i)* static heating and *ii)* dynamic heating. For static heating, at least two of the three attributes are fixed. (**FIS** refers to **fixed iterations** and population **size** per parameter; **FTS** refers to **fixed time** and population **size** per parameter; **FIT** refers to **fixed iterations** and fixed **time** per parameter.) For dynamic heating, at least two attributes are variable. (**VIT** refers to **variable iterations** and **time** per parameter; **VIS** refers to **variable iterations** and population **size**; **VTS** refers to **variable time** and population **size**.) In our experiments, we will only consider the FIS, FTS, and VIT strategies.

4.1 Static Heating

Static heating means that at least two of the above three parameters are fixed and identical for all PLSA parameter values considered during the optimization process. As a consequence, the third parameter is either given as well or can be calculated before run-time for each PLSA parameter value separately. As illustrated in Figure 3 on the left, there are four possible static heating schemes.

4.1.1 PLSA Parameter Fixed — Standard Hybrid Approach

Fixing all three parameters is identical to keeping p constant. Thus, only a single PLSA parameter value is used during the optimization process. This scheme represents the common way to incorporate PLSAs into GSAs and is taken as the reference for the other schemes as actually no heating is performed.

4.1.2 Number of Iterations and Size of Solution Candidate Set Fixed Per PLSA Parameter (FIS)

In this strategy (FIS), the parameter p_i is constant for exactly $t_i = t_p$ iterations. The question is, therefore, how many iterations t_p may be performed per parameter within the time budget T_{max}. Having the constraint

$$T_{max} \geq t_p N(C_{fix} + C(p_1)) + t_p N(C_{fix} + C(p_2)) + \ldots + t_p N(C_{fix} + C(p_n))$$

we obtain t_p with

$$t_p = \left\lfloor \frac{T_{max}}{N \sum_{i=1}^{n} (C_{fix} + C(p_i))} \right\rfloor \tag{2}$$

as the number of iterations assigned to each p_i.

4.1.3 Amount of Time and Size of Solution Candidate Set Fixed Per PLSA Parameter (FTS)

For the FTS strategy, the points in time where p is increased are equi-distant and may be simply computed as follows. Obviously the time budget, when equally split between n parameters, becomes $T_p = T_{max}/n$ per parameter. Hence, the number of iterations t_i that may be performed using parameter p_i, $i = 1, \ldots, n$ is restricted by

$$t_i N(C_{fix} + C(p_i)) \le T_p, \forall i = 1, \ldots, n$$

Thus, we obtain

$$t_i = \left\lfloor \frac{T_{max}}{nN(C_{fix} + C(p_i))} \right\rfloor \tag{3}$$

as the maximum number of iterations that may be computed using parameter p_i in order to stay within the given time budget.

4.1.4 Number of Iterations and Amount of Time Fixed Per PLSA Parameter (FIT)

With the FIT scheme the size of the solution candidate set is different for each PLSA parameter considered. The time per iteration for parameter p_i is given by $T_i = T_{max}/t_{max}$ and is the same for all p_i with $1 \le i \le n$. This relation together with the constraint

$$T_i \ge N_i(C_{fix} + C(p_i))$$

yields

$$N_i = \left\lfloor \frac{T_{max}}{t_{max}(C_{fix} + C(p_i))} \right\rfloor \tag{4}$$

as the maximum size of the solution candidate set for p_i.

4.2 Dynamic Heating

In contrast to static heating, dynamic heating refers to the case in which at least two of the three optimization parameters are not fixed and may vary for different PLSA parameters. The four potential types of dynamic heating are shown in Figure 3. However, the scenario where all three optimization parameters are variable and may be different for each PLSA parameter is more hypothetical than realistic. This approach is not investigated in this paper and only listed for reasons of completeness. Hence, we consider three dynamic heating schemes where only one parameter is fixed. One of the variable parameters is determined dynamically during run-time according to a predefined criterion. Here, the criterion is whether an improvement with regard to the solutions generated can be observed during a certain time interval (measured in seconds, number of solutions generated, or number of iterations performed). The time constraint is defined in terms of the remaining variable parameter.

4.2.1 Number of Iterations and Size of Solution Candidate Set Variable Per PLSA Parameter (VIS)

With the VIS strategy, the time $T_i = T_{max} / n$ per PLSA parameter value is fixed (and identical for all p_i). If the time constraint is defined on the basis of the number of solutions generated, the hybrid works as follows: As long as the time T_i is not exceeded, new solutions are generated using p_i and copied to the next solution candidate set—otherwise, the next GSA iteration with p_{i+1} is performed. If, however, the time elapsed for the current iteration is less than T_i *and* none of the recently generated N_{stag} solutions achieves an improvement in fitness, the next iteration with p_i is started.

It is not practical to consider a certain number of iterations as the time constraint—since the time per iteration is not known, there is no condition that determines when the filling of the next solution candidate set can be stopped.

4.2.2 Amount of Time and Size of Solution Candidate Set Variable Per PLSA Parameter (VTS)

There are two heating schemes possible when the number of iterations t_i per PLSA parameter is a constant value $t_i = t_{max} / n$. One scheme we call VTS-S, in which the next solution candidate set is filled with new solution candidates until, for N_{stag} solutions, no improvement in fitness is observed. In this case the same procedure is applied to the next iteration using the same parameter p_i. If t_i iterations have been performed for p_i, the next PLSA parameter p_{i+1} is taken.

In the other heating scheme, which we call VTS-T, the filling of the next solution candidate set is stopped if, for T_{stag} seconds, the quality of the best solution in the solution candidate set has stagnated (i.e. has not improved).

4.2.3 Number of Iterations and Amount of Time Variable Per PLSA Parameter (VIT)

Here again there are two possible variations. The first, called VIT-I, considers the number of iterations as the time constraint. The next PLSA parameter value is taken when for a number t_{stag} of iterations the quality of the best solution in the solution candidate set has not improved. As a consequence, for each parameter a different amount of time may be considered until the stagnation condition is fulfilled.

The alternative VIT-T is to define the time constraint in seconds. In this case, the next PLSA parameter value is taken when, for T_{stag} seconds, no improvement in fitness was achieved. As a consequence, for each parameter a different number of iterations may be considered until the stagnation condition is fulfilled.

4.3 Simulated Heating Applied to Binary Knapsack Problem

In order to further illuminate simulated heating, we begin by demonstrating the technique on a widely known problem, namely the binary (0-1) knapsack problem (KP). This problem has been studied extensively, and good exact solutions methods for it have been developed (e.g. see [31]). The exact solutions are based on either branch-and-bound or dynamic programming techniques. In this problem, we are given a set of n items, each with profit Δ_j and weight w_j, which must be packed in a knapsack with weight capacity c. The problem consists of selecting a subset of the n items whose total weight does not exceed c and whose total profit is a maximum. This can be expressed formally as:

$$\text{maximize } z = \sum_{j=1}^{n} \Delta_j x_j \tag{5}$$

$$\text{subject to } \sum_{j=1}^{n} w_j x_j \le c \tag{6}$$

$$x_j \in \{0, 1\}, j \in \{1, \ldots, n\} \tag{7}$$

where $x_j = 1$ if item j is selected, and $x_j = 0$ otherwise.

Balas and Zemel [1] first introduced the “core problem” as an efficient way of solving KP, and most of the exact algorithms have been based on this idea. Pisinger [30] has modeled the hardness of the core problem and noted that it is important to test at a variety of weight capacities. He proposed a series of randomly generated test instances for KP. In our experiments we generate test instances using the test generator function described in appendix B of [30]. We compare our results to the exact solution described in [31], for which the c-code can be found at [40].

4.3.1 Implementation

To solve the KP we use a GSA/PLSA hybrid as discussed in section 3 where an evolutionary algorithm is the global search algorithm (GSA) and a simple pairwise exchange is the parameterized local search algorithm (PLSA). The evolutionary algorithm and local search are explained below.

GSA: Evolutionary Algorithm

Each candidate solution s is encoded as a binary vector $\vec{x}$, where x_j are the binary decision variables from equation (7) above. The weight of a given solution candidate s is $w_s = \sum_{j=1}^{n} x_j w_j$, and the profit of s is $\Delta_s = \sum_{j=1}^{n} x_j \Delta_j$. The sum of the profits of all items is defined as $\Delta_t = \sum_{j=1}^{n} \Delta_j$. We define a fitness function for s which we would like to *minimize*:

$$F(s) = \begin{cases} \Delta_t - \Delta_s & \text{if } w_s \le c \\ \Delta_t + w_s & \text{if } w_s > c \end{cases}. \tag{8}$$

Thus we penalize solution candidates whose weight exceeds the capacity, and seek to maximize the profit. The Δ_t term was added so that $F(s)$ is never negative. For the KP experiments we used a standard simple genetic algorithm described in [15] with one point crossover, crossover probability 0.9, non-overlapping populations of size $\text{popsize} = 100$, and elitism.

Parameterized Local Search for Knapsack Problem

At the beginning of the optimization algorithm, the items are sorted by increasing profit , so that $\Delta_i \le \Delta_j$ for all $i < j$. Given an input solution candidate s, the local search first computes its weight w_s. If $w_s > c$, items are removed (x_i set to zero) starting at $i = 0$ until $w_s \le c$. For local search parameter $p = 1$, this is the only operation performed. For $p > 1$, pair swap operations are also performed as explained in Figure 4, where we attempt to replace an item from the solution candidate with a more profitable item not included in the solution candidate. The number of such pair swap operations is p. Thus the local search algorithm requires more computation time and searches the local area more thoroughly for higher p. These are the monotonicity requirements expressed in Equation 1. We define parameter $p = 0$ as no local search—i.e. the optimization is an evolutionary algorithm only, and no local

```
Pair Swap Local Search() {
    Input: solution candidate $s_{\text{in}}$ of size $n$, fitness function $F$
    Output: new solution candidate $s_{\text{out}}$
    $i = n - 1$
    $s_{\text{out}} = s_{\text{in}}$
    $\text{bestScore} = F(s_{\text{in}})$
    $\text{count} = 0$
    while $((i > 0)$ and $(\text{count} < pn))$ {
        $j = 0$
        while $((j < i)$ and $(\text{count} < pn))$ {
            if $(s_{\text{out}}[i] \neq s_{\text{out}}[j])$ {
                $\text{temp} = s_{\text{out}}[i]$
                $s_{\text{out}}[i] = s_{\text{out}}[j]$
                $s_{\text{out}}[j] = \text{temp}$
                $\text{score} = F(s_{\text{out}})$
                if $(\text{score} < \text{bestScore})$ {
                    $\text{bestScore} = \text{score}$
                }
                else {
                    $\text{temp} = s_{\text{out}}[i]$
                    $s_{\text{out}}[i] = s_{\text{out}}[j]$
                    $s_{\text{out}}[j] = \text{temp}$
                }
                $\text{count} = \text{count} + 1$
            }
            $j = j + 1$
        }
        $i = i - 1$
    }
}
```

Figure 4. Pseudo-code for pair swap local search for binary knapsack problem.

search is performed.

4.3.2 Influence of p on the PLSA run-time and accuracy

To test the binary knapsack problem, we generated 1000 pseudo-random test instances for each technique as suggested in [30]. The weights and profits in these instances were strongly correlated. The weight capacity c_i of the ith instance is given by $c_i = \lfloor (iW)/1001 \rfloor$ where W is the sum of the weights of all items. For each test instance we compared the hybrid solution with an exact solution the problem using the method given in [31]. We defined an error sum over all the problem instances as a figure of merit for the hybrid solution technique:

$$\varepsilon = \sum_{i=1}^{1000} (\alpha_i - \beta_i) \tag{9}$$

where α_i is the profit given by the exact solution and β_i is the profit given by the hybrid solution.

Figure 5(a) shows how the run-time of the pair swap PLSA increases with p. Figure 5(b) depicts the sum of errors (Equation 9) for the binary knapsack problem for different values of p with the number of generations fixed at 10. We can see that higher values of p produce smaller error, at the expense of increased run time. Thus the pair swap PLSA satisfies the monotonicity requirement from Equation 1.

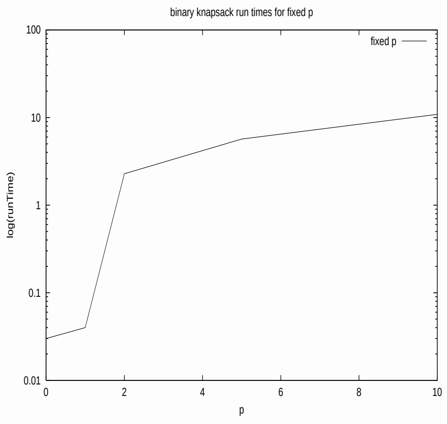


Figure 5. (a) **Local search run times** vs. p for **binary knapsack**.

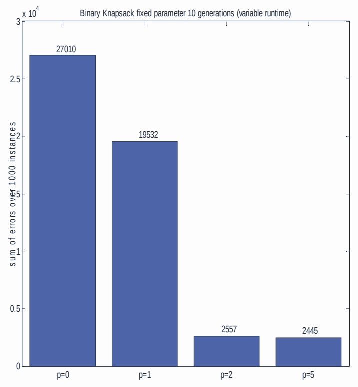


Figure 5. (b) **Standard hybrid approach** for binary knapsack (fixed p, no heating) using a **fixed number of generations** and not fixing overall hybrid run time. **Cumulative error** shown for hybrids utilizing different p. Higher p is more accurate but requires longer run times.

4.3.3 Standard Hybrid Approach (Fixed PLSA Parameter)

The standard approach to hybrid global/local searches is to run the local search at a fixed parameter. This is shown in Figure 6 below for different values of p and for two different run times. The y-axis in the figure corresponds to the sum of errors over all test cases (equation 9). We see that, for a fixed optimization run-time, the optimal value of local search parameter p using the standard hybrid approach can depend on the run-time and data input—for a run time of 2 seconds, the best value of p is 2, while for a run time of 5 seconds, the best value of p is 5. We note here and with the other applications studied that this value of p cannot be predicted in advance.

4.3.4 Static Heating Schemes

The static heating schemes FIS and FTS were performed for the binary knapsack problem. Results are shown in Figure 7 for run times of 1 and 5 seconds, and compared with the standard hybrid approach for different val-

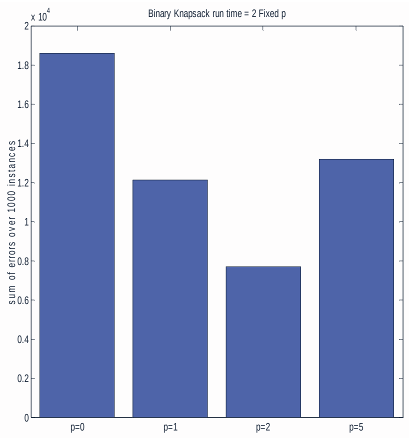


Figure 6. (a) **Standard hybrid approach** applied to **binary knapsack** for different values of p, where p is fixed throughout. Y-axis is sum of errors. Run time is **2 seconds**.

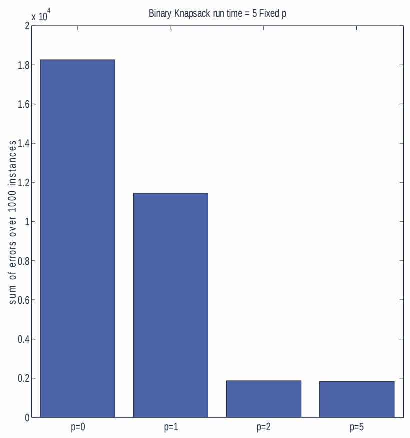


Figure 6. (b) **Standard hybrid approach** applied to **binary knapsack** for different values of p, where p is fixed throughout. Y-axis is sum of errors. Run time is **5 seconds**

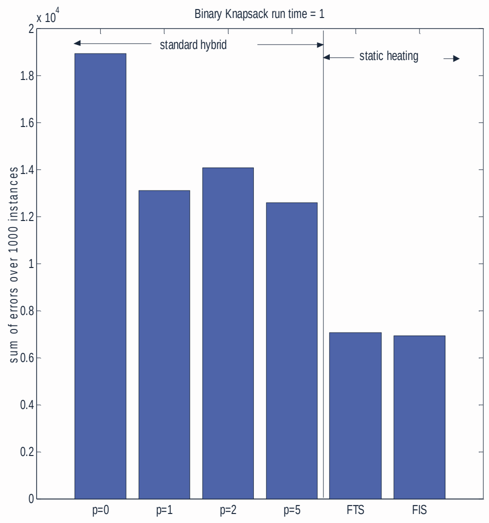


Figure 7. (a)**Static heating** (2 bars on right) applied to **binary knapsack compared to the standard hybrid approach** (4 bars on left). Y axis is sum of errors over all 1000 problem instances. The 4 bars on left correspond to the standard hybrid approach. Run time is **1 second**.

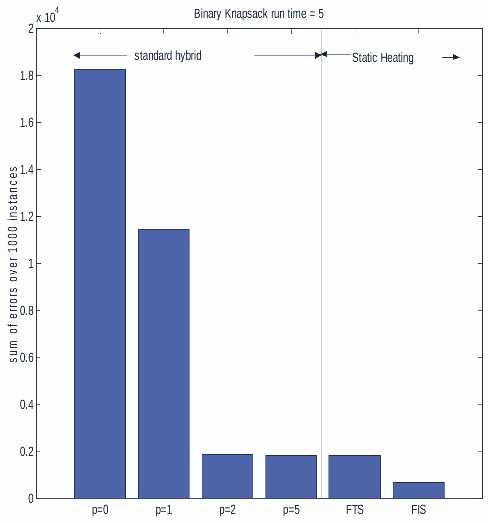
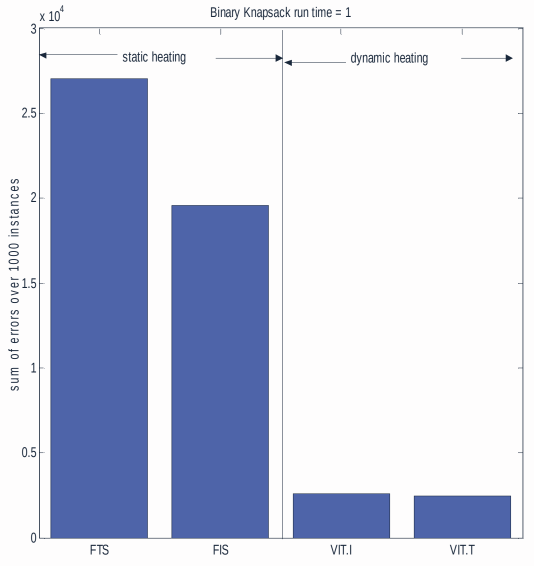


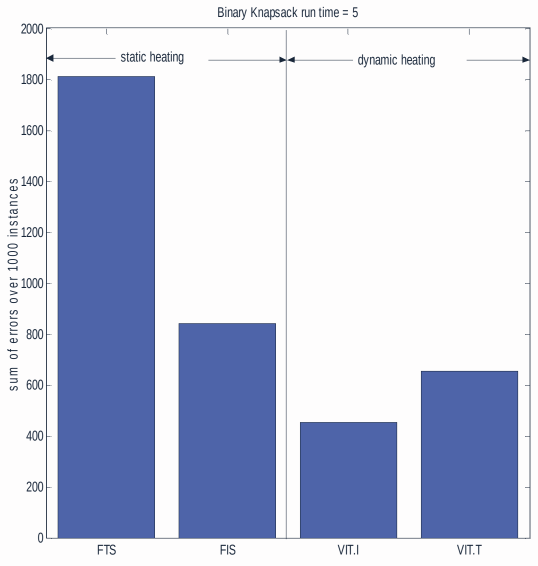
Figure 7. (b)**Static heating** (2 bars on right) applied to **binary knapsack compared to the standard hybrid approach** (4 bars on left). Y axis is sum of errors over all 1000 problem instances. The 4 bars on left correspond to the standard hybrid approach. Run time is **5 seconds**.

ues of p. It can be seen that the static heating scheme outperformed the standard hybrid approach, and that this improvement is greater for the shorter run times.

4.3.5 Dynamic Heating Schemes

The dynamic heating schemes VIT.I and VIT.T were performed for the binary knapsack application. Recall that VIT stands for variable iterations and time per parameter; during the optimization the next PLSA parameter is taken when, for a given number of iterations (VIT.I) or a given time (VIT.T), the quality of the solution candidate has not improved. Figure 8 shows results for these dynamic schemes. Results for static heating schemes.are shown on the





(a) **(b)**

Figure 8. **Dynamic heating** for **binary knapsack** (two bars on right) **compared to static heating** (two bars on left). VIT refers to variable iterations and time per parameter, with the next parameter taken if, for a given number of iterations (VIT.I) or a given time (VIT.T), the solution has not improved. Run time is **1 second (a)** or **5 seconds (b)**. Y axis is cumulative error over all problem intances (note the different y scales for the two plots.

right of the figure for comparison. We observe that the dynamic heating schemes outperform the static heating schemes significantly, and that the amount of improvement is greater for shorter run times.

5 Embedded Systems Applications

Next we will demonstrate our simulated heating technique on two problems in the design of embedded systems. For many problems in system design, the user wishes to first quickly evaluate many trade-offs in the system, often in an interactive environment, and then to refine a few of the best design points as thoroughly as possible. Often, an exact system simulation may take days or weeks. In this context, it is quite useful to have optimization techniques where the run-time can be controlled, and which will generate a solution of maximum quality in the allotted time.

Hybrid global/local search techniques are most effective in problems with complicated search spaces, and problems for which local search techniques have been developed that make maximum use of problem-specific information. We investigate the effectiveness of the simulated heating approach on two such applications in electronic design, namely software optimization in embedded systems and voltage scaling for multiprocessors. These problems are very different in structure, but both have vast and complicated solution spaces. The parameterized local search algorithms (PLSAs) for these applications exhibit a wide range of accuracy/complexity trade-offs. We will give a brief overview of each application so that the reader can get a flavor of these trade-offs. More details about the implementation for these two problems are given in the Appendix.

5.1 Memory Cost Minimization Application

5.1.1 Background

Digital signal processing (DSP) applications can be specified as a dataflow graph [5]. In dataflow, a computational specification is represented as a directed graph in which vertices (*actors*) specify computational functions of arbitrary complexity, and edges specify FIFO communication between functions.

A *schedule* for a dataflow graph is simply a specification of the order in which the functions should execute. A given DSP application can be accomplished with a variety of different schedules—we would like to find a schedule which minimzes the memory requirement. A *periodic schedule* for a dataflow graph is a schedule that invokes each actor at least once and produces no net change in the number of data items queued on each edge. A software synthesis tool generates application programs from a given schedule by piecing together (*inlining*) code modules from a predefined library of software building blocks associated with each actor. The sequence of code modules and subroutine calls that is generated from a dataflow graph is processed by a buffer management phase that inserts the necessary target program statements to route data appropriately between actors.

The scheduling phase has a large impact on the memory requirement of the final implementations, and it is this memory requirement we wish to minimize in our optimization. The key components of this memory requirement are the code size cost (the sum of the code sizes of all inlined modules, and of all inter-actor looping construct), and the buffering cost (the amount of memory allocated to accommodate inter-actor data transfers). Even for a simple dataflow graph, the underlying range of trade-offs may be very complex. We denote a *schedule loop* with the notation $(nT_1T_2\ldots T_m)$, which specifies the successive repetition n times of a subschedule $T_1T_2\ldots T_m$, where the T_i are actors. A schedule that contains zero or more schedule loops is called a *looped schedule*, and a schedule that contains exactly zero schedule loops is called a *flat schedule* (thus, a flat schedule is a looped schedule, but not vice-versa).

Consider two schedules $S_1 = (8YZ)X(2YZ)$ and $S_2 = X(10YZ)$ which repeat the actors X, Y, and Z the same number of times (1, 10, 10, respectively). The *code size costs* for schedules S_1 and S_2 can be expressed, respectively, as $\kappa(X) + \kappa(Y) + \kappa(Z) + L_c$, where L_c denotes the processor-dependent, code size overhead of a software looping construct, and $\kappa(A)$ denotes the program memory cost of the library code module for an actor A. The code size of schedule S_1 is larger because it contains more "actor appearances" than schedule S_2 (e.g., an actor Y appears twice in S_1 vs. only once in S_2), and S_1 also contains more schedule loops (2 vs. 1). The *buffering cost* of a schedule is computed as the sum over all edges e of the maximum number of buffered (produced, but not yet consumed) tokens that coexist on e throughout execution of the schedule. Thus, the buffering costs of S_1 and S_2 are 11 and 19, respectively. The *memory cost* of a schedule is the sum of its code size and buffering costs. Thus, depending on the relative magnitudes of $\kappa(X)$, $\kappa(Y)$, $\kappa(Z)$, and L_c, either S_1 or S_2 may have lower memory cost.

5.1.2 MCMP Problem Statement

The *memory cost minimization problem* (*MCMP*) is the problem of computing a looped schedule that minimizes the memory cost for a given dataflow graph, and a given set of actor and loop code sizes. It has been shown that this problem is NP-complete [5]. We have previously described a tractable algorithm called *CDPPO* (code size dynamic programming post optimization) [4,38,39] which can be used as a local search for MCMP. As explained in the Appendix, the CDPPO algorithm can be formulated naturally as a PLSA with a single parameter such that accuracy and run-time both increase *monotonically* with the parameter value. In our previous work combining CDPPO with an evolutionary algorithm we uniformly applied the "full-strength" (maximum accuracy/maximum run-time) form of CDPPO, and as conventionally done with local search techniques, did not explore application of its PLSA form. Our objective with CDPPO in this work is very different: we seek to understand the effectiveness of its PLSA formulation, and the manner in which the associated *monotonic accuracy/run-time* trade-off should be managed during optimization.

5.2 Multiprocessor Voltage Scaling Application

5.2.1 Background

Dynamic voltage scaling [26] in microprocessors is an important advancing technology. It allows the average power consumption in a device to be reduced by slowing down (by lowering the voltage) some tasks in the application. The application is again specified as a dataflow graph. We are given a schedule (ordering of tasks on the processors) and a constraint on the throughput of the system. We wish to find a set of voltages for all the tasks that minimize the average power of the system while satisfying the throughput constraint. The only way to compute the throughput exactly in these systems is via a full system simulation. However, simulation is computationally intensive and we would like to mimimze the number of simulations required during synthesis. We have previously demonstrated that a data structure, called the *period graph*, can be used as an efficient estimator for the system throughput [2] and thus reduce the number of simulations required.

5.2.2 Using the Period Graph for Local Search

As explained in [2], we can estimate the throughput of the system as voltage levels are changed by calculating the maximum cycle mean[4] (MCM) [23] of the period graph. In order to construct the period graph, we must perform one full system simulation at an initial point—after the period graph is constructed we may use the MCM estimate without re-simulating the system. It is shown in [2] that the MCM of the period graph is an accurate estimate for the throughput if the task execution times are varied around a limited region (local search), and that the quality of the estimate increases as the size of this region decreases. A variety of efficient, low polynomial-time algorithms have been developed for computing the maximum cycle mean (e.g., see [10]).

We can use the size of the local search neighborhood as the parameter p in a parameterized local search algorithm (PLSA). We call this parameter the resimulation threshold (r), and define it as the vector distance between a candidate point (vector of voltages) and the voltage vector V from which the period graph was constructed. To search around a given point V in the design space, we must simulate once and build the period graph. Then, as long as the local search points are within a distance r from V, we can use the (efficient) period graph estimate. For points outside r, we must resimulate and rebuild the period graph. Consequently, there is a trade-off between speed and accuracy for r—as r decreases, the period graph estimate is more accurate, but the local search is slower since simulation is performed more often.

5.2.3 Voltage Scaling Problem Statement

We assume that a schedule has been computed beforehand so that the ordering of the tasks on the processors

4. Here the maximum cycle mean is the maximum, over all directed cycles of the period graph, of the sum of the task execution times on a cycle divided by the sum of the edge delays (initial tokens) on a cycle.

is known. The optimization problem we address consists of finding the voltage vector $V = (v_1, v_2, \ldots, v_n)$ for the n tasks in the application graph, such that the energy per computation period (average power) is minimized and the throughput satisfies some pre-specified constraint (e.g., as determined by the sample period in a DSP application). For each task, as its voltage is decreased, its energy is decreased and its execution time is increased, as described in [2]. The computation period is determined from the period graph. A simple example is shown in Figure 9. Here we can see that by decrasing the voltage on task B, the average power is reduced. There is a potentially vast search space for many practical applications. For example, if we consider discrete voltage steps of 0.1 volts over a range of 5 volts, there are n^{50} possible voltage vectors V from which to search. The number of tasks n in an application may be in the hundreds.

5.3 Experiments

In this section we present experiments designed to examine several aspects of simulated heating for the two embedded systems applications. We would like to know how simulated heating compares to the standard hybrid technique of using a fixed parameter (fixed p). We summarize the fixed p results for all problems for different values of p. We examine how the optimal value of p for the standard hybrid method depends on the application.

Next we compare both the static and dynamic heating schemes to the standard approach, and to each other. For the static heating experiments, we utilize the FIS and FTS strategies. Recall that **FIS** refers to **fixed** number of

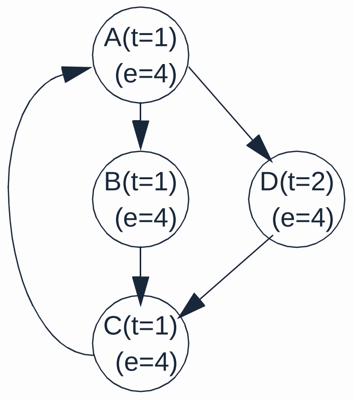


Figure 9. (a) Period Graph before voltage scaling. The numbers represent execution times (t) and energies (e) of the tasks. The execution period is determined by the longest cycle, A-D-C, whose sum of execution times is 4 units. The energy of each task is 4 units. The average power is 4 units (16 total energy divided by period of 4).

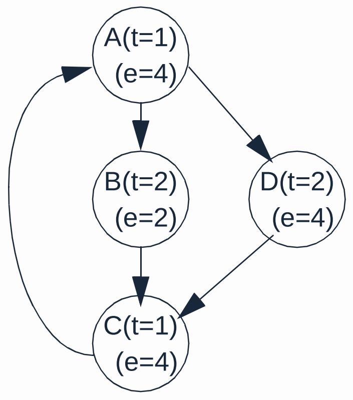


Figure 9. (b) After voltage scaling. The voltage on task B has been reduced, increasing its execution time from 1 unit to 2 units and decreasing its energy consumption from 4 units to 2 units. The overall execution period is still 4 units since both cycles A-D-C and A-B-C now have execution time of 4. The average power is 3.5 units (14 total energy divided by 4 period of 4).

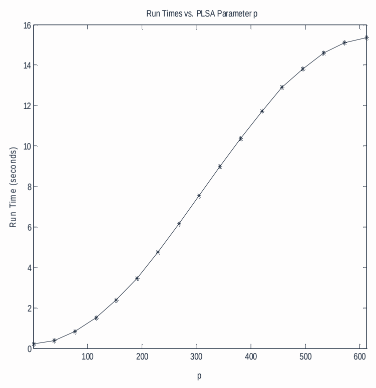


Figure 10. (a) **Local search run times** vs. p for memory cost minimization **(MCMP)** application.

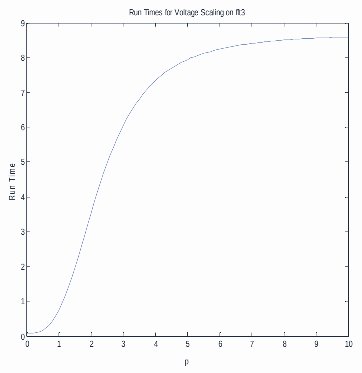


Figure 10. (b) **Local search run times** vs. p for **voltage scaling** application.

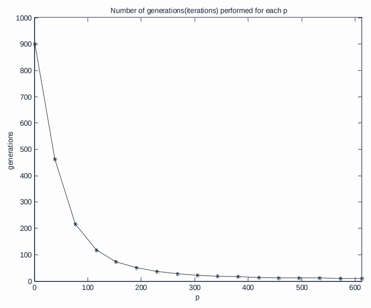


Figure 10. (c) **Standard hybrid approach** (fixed p, no heating), MCMP application, using a **fixed run time**. Number of generations completed is shown for hybrids utilizing different values of p. Fewer generations are completed for higher p.

iterations and population **size** per parameter, and **FTS** refers to **fixed time** and population **size** per parameter. For the dynamic heating experiments, we utilize the two variants of the **VIT** strategy (**variable iterations** and **time** per parameter). We also examine the role of parameter range and population size on the optimization results.

5.4 Results

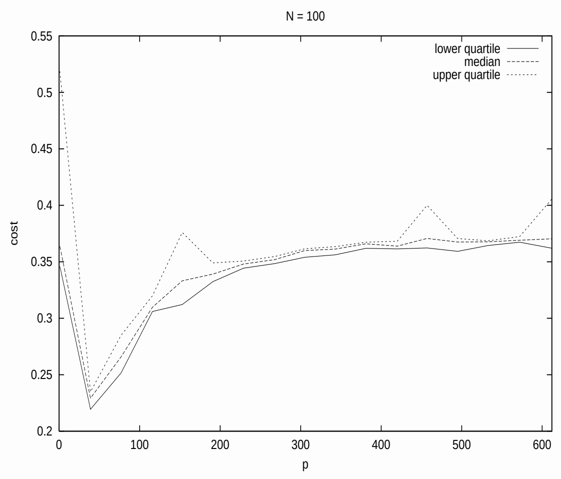
5.4.1 Influence of p on the PLSA run-time and accuracy

Recall that there is a tradeoff between accuracy and run-time for the PLSA. Lower values of local search parameter p mean the local search executes faster, but is not as accurate. Figures 10(a)-(b) show how the run-time of the parameterized local search (PLSA) varies for the two applications. It can be seen that the monotonicity property, Equation 1, is satisfied for the PLSAs.

5.4.2 Standard Hybrid Approach (Fixed PLSA Parameter)

The standard approach to hybrid global/local searches is to run the local search at a fixed parameter. We present results for this method below. It is important to note that, for a fixed optimization run-time, the optimal value of local search parameter p can depend on the run-time and data input and cannot be predicted in advance.

Figures 11(a) and 11(b) show results for the MCMP optimization using fixed values of p (standard



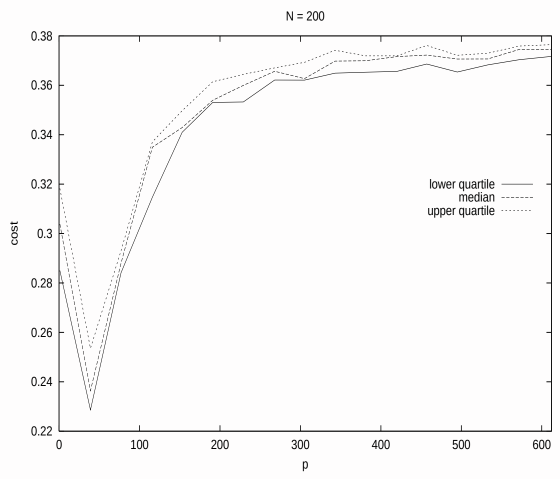


Figure 11. (a)**Standard hybrid approach** MCMP application using fixed PLSA parameter p , Hybrid was run for 5 hours at each value of p. Population size in GA was N=100. Median, lower quartile, and upper quartile of 11 different runs shown in the three curves for each p (Lower memory cost is better).

Figure 11. (b) Same as (a) but population size increased to N=200.

approach—no heating), for 11 different initial populations, for population sizes $N = 100, 200$. The y-axis on these graphs corresponds to the memory cost of the optimized schedule (section 5.1.1) so that lower values are better. The x-axis corresponds to the fixed p value. For each value of p , the hybrid search was run for a time budget of 5 hours with a fixed value of p . The same set of initial populations was used. From these graphs, it can be seen that the local search performs best for values of p around 39. Figure 10(c) shows the number of iterations (generations in the GSA) performed for each value of p . As p increases, fewer generations can be completed in the fixed optimization run time.

Figures 12(a) and 12(b) show results for the voltage scaling on 6 different input dataflow graphs, for fixed values of p (no heating), for 11 different initial populations, using both hill climb and Monte Carlo local search methods. For each value of p , the hybrid search was run for a time budget of 20 minutes with a fixed value of p . The y-axis on the graph corresponds to the ratio of the optimized average power to the initial power, so that lower values are better. For each p , the same set of initial populations was used. From these graphs, it can be seen that the best value of p may also depend on the specific problem instance.

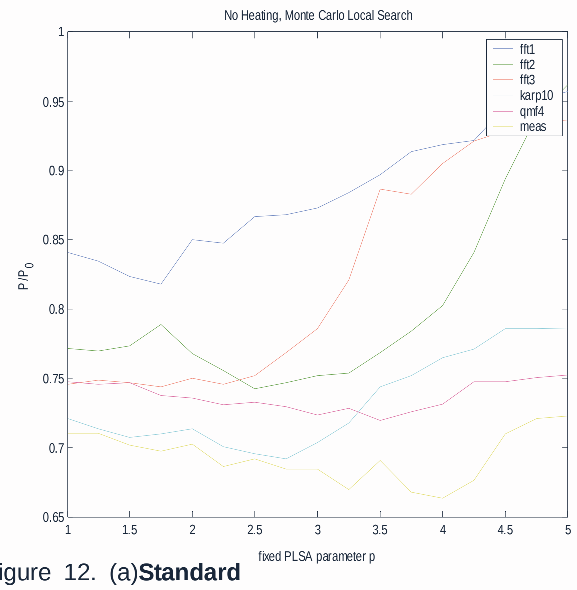


Figure 12. (a)**Standard hybrid approach** using fixed PLSA parameters, **voltage scaling application**, with **Monte Carlo local search**. Hybrid was run for 20 minutes at each value of p. Median of 11 runs for each p. Lower values of power are better. We see that the optimal value of p is different for the six different input dataflow graphs.

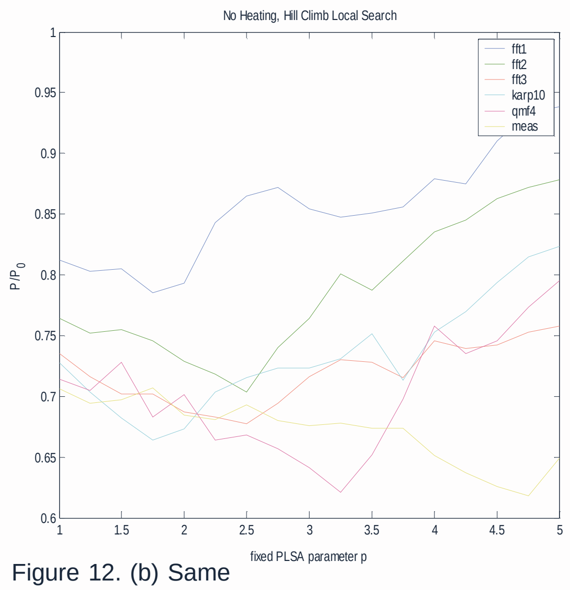


Figure 12. (b) Same as (a) but with **hill climb local search**.

5.4.3 Static Heating Schemes

For the MCMP application, the run-time limit for the hybrid was set to $T_{max} = 5\text{hours}$. Two sets of PLSA parameters were used, $R^1 = [1, 153, 305, 457, 612]$ and $R^2 = [1, 39, 77, 116, 153]$. The value $p = 612$ corresponds to the total number of actor invocations in schedule for the MCMP application and is thus the maximum (highest accuracy) possible. The parameter set R^2 was chosen so that it is centered around the best fixed p values. Figure 13 summarizes the results for the MCMP application with GSA population size $N = 100$. In Figure 13, eleven runs were performed for each heating scheme and for each parameter set. The box plot[5] Figure 13(a) corresponds to FIS with parameter set R^1. Figure 13(b) corresponds to FIS with parameter set R^2. Figure 13(c) corresponds to FTS with parameter set R^1. Figure 13(d) corresponds to FTS with parameter set R^2. The solid curves in the figure are the results for fixed p.

For the voltage scaling application, $T_{max} = 20\text{minutes}$. For FIS and FTS, the parameter sets used were

5. The ‘box’ in the box plot stretches from the 25th percentile (‘lower hinge’) to the 75th percentile (‘upper hinge’). The median is shown as a line across the box. The ‘whisker’ lines are drawn at the 10th and 90th percentiles. Outliers are shown with a ‘+’ character.

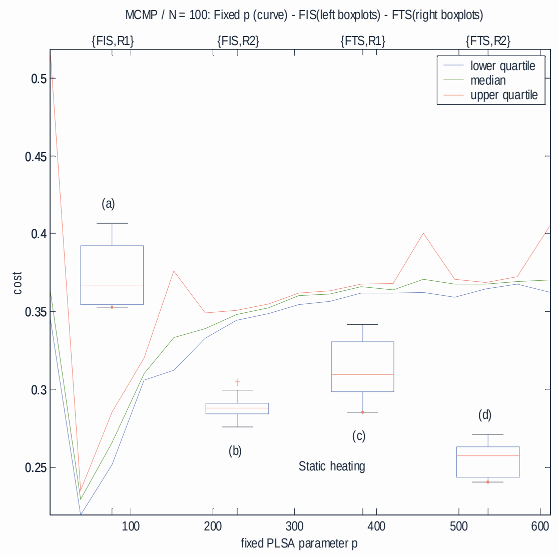


Figure 13. **Static heating** for MCMP with the local search parameter p varied in two different ranges—the first range covers all possible values (1-612), while the second range (1-153) is concentrated around the best fixed p value. $(a)\{FIS, R^1\}$, $(b)\{FIS, R^2\}$, $(c)\{FTS, R^1\}$, $(d)\{FTS, R^2\}$. The solid curve depicts the standard hybrid approach for different values of p. Lower values of cost are better. The box plots display the static heating results. The solid line across the box represents the median over all calculations. The lowest cost is obtained for the standard hybrid approach with p=39. The best static heating scheme is (d), corresponding to FTS operating in the restricted parameter range which includes p=39. We note that this value of p could not be determined in advance, and could only be found by running the standard hybrid solution for all values of p.

$R^3 = [1, 2, 3, 4, 5]$ and $R^4 = [2.25, 2.5, 2.75, 3, 3.25]$. The parameter set R^3 was chosen by examining the fidelity of the period graph estimator. Recall that the PLSA parameter p is related to the resimulation threshold. It is observed that for $p < 1$ the fidelity of the estimator is poor. For p greater than 5, with the voltage increments used, the resimulation threshold is so small that simulation is done almost every time. This corresponds to the highest accuracy setting. The parameter set R^4 was chosen to center around the best fixed p values. Results for FIS and FTS on the fft2 application using the Monte Carlo local search are shown in Figure 14. Table 1 summarizes the iterations performed for each parameter for both FIS and FTS with both parameter ranges.

heating scheme		iterations per parameter p							
type	range	1	39	77	115	153	305	457	612
FIS	[1,612]	4	x	x	x	4	4	4	4
FIS	[1,153]	33	33	33	33	33	x	x	x
FTS	[1,612]	176	x	x	x	14	4	2	2
FTS	[1,153]	175	94	42	23	14	x	x	x

Table 1. Iterations performed per parameter value for four different heating schemes for MCMP. The numbers correspond to a single optimization run; for the other ten runs they look slightly different.

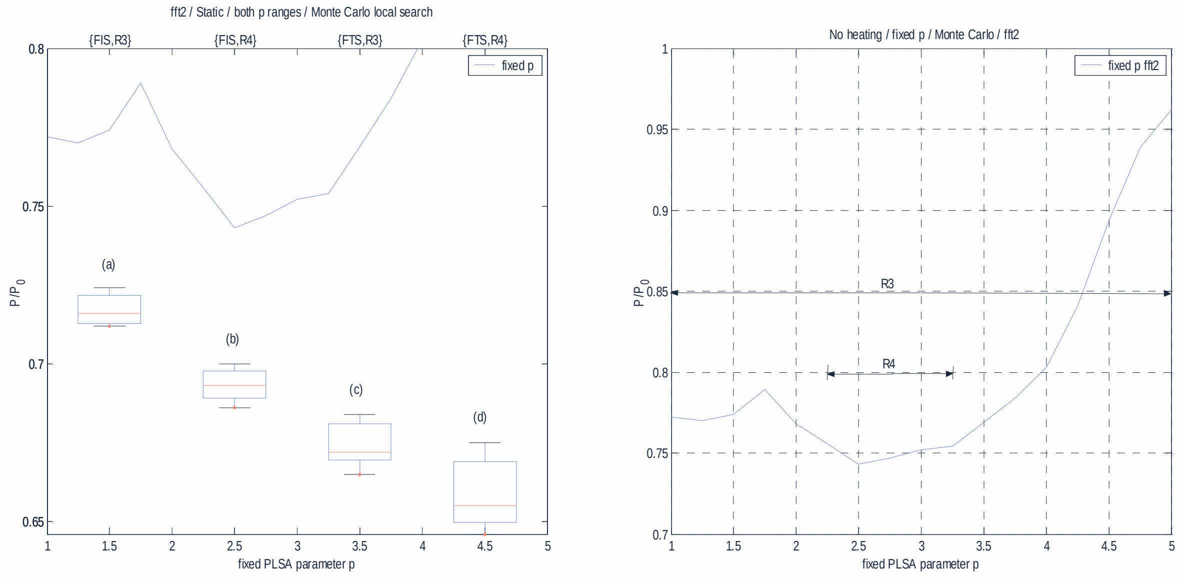


Figure 14. **Static heating** for **voltage scaling** with different parameter ranges— $(a)\{FIS, R^3\}$, $(b)\{FIS, R^4\}$, $(c)\{FTS, R^3\}$, $(d)\{FTS, R^4\}$ (shown in the four box plots) compared with the standard hybrid method results (fixed values of *p* shown in the solid line). Here the static heating schemes all perform better than the standard hybrid approach. The first parameter range includes all values of p, while the second range is centered around the best fixed p value. This is shown in more detail on the right.

5.4.4 Dynamic Heating Schemes

The dynamic heating schemes VIT.I and VIT.T were performed for both the MCMP and the voltage scaling applications. Recall that VIT stands for variable iterations and time per parameter; during the optimization the next PLSA parameter is taken when, for a given number of iterations (VIT.I) or a given time (VIT.T), the quality of the solution candidate has not improved.

For the MCMP application, the run-time limit for the hybrid was set to $T_{\max} = 5\text{hours}$ and the same two sets of PLSA parameters were used as for the static heating case. Eleven runs were performed for all cases. Results for dynamic heating on the MCMP application are shown in Figure 15. For the voltage scaling application, $T_{\max} = 20\text{minutes}$. Results for voltage scaling with VIT.I and VIT.T using the Monte Carlo local search are shown in Figure 16. For the dynamic heating schemes, the search algorithm operates with a given PLSA parameter until the quality of the best solution has not improved for either t_{stag} iterations (VIT.I) or T_{stag} seconds (VIT.T). It is therefore interesting to observe the amount of time spent on each parameter during the optimization. This is illustrated in Figure 17.

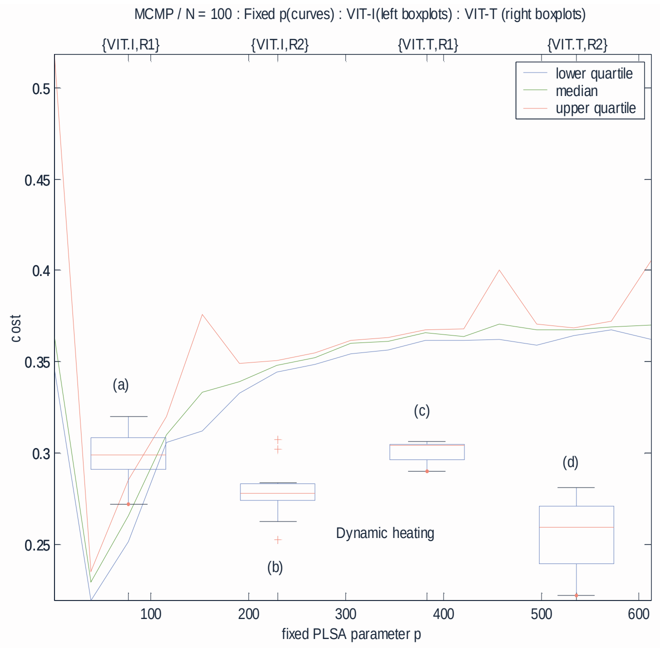
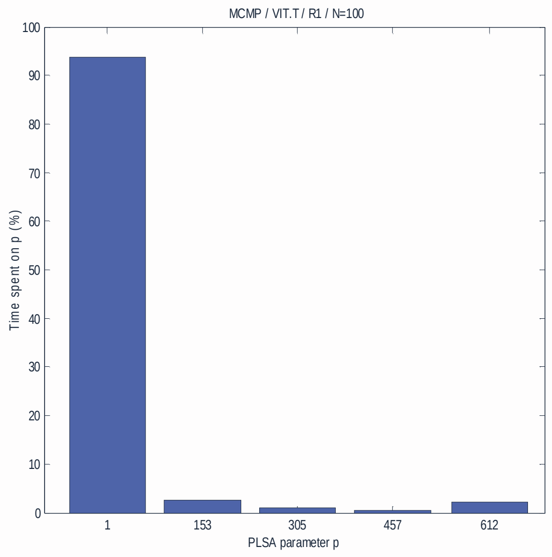


Figure 15. **Dynamic heating** for **MCMP** with different parameter ranges—the four box plots depict the two variants over the two ranges—$(a)\{\text{VIT.I}, R^1\}$, $(b)\{\text{VIT.I}, R^2\}$, $(c)\{\text{VIT.T}, R^1\}$, $(d)\{\text{VIT.T}, R^2\}$. The solid line represents the standard hybrid technique with p fixed at different values from 1 to 612. The solid lines across the boxes represents the median over all calculations. The lowest cost is obtained for the standard hybrid approach with p=39. The best dynamic heating scheme is (d), corresponding to VIT.T operating in the restricted parameter range which includes p=39. We note that this value of p could not be determined in advance, and could only be found by running the standard hybrid solution for all values of p.



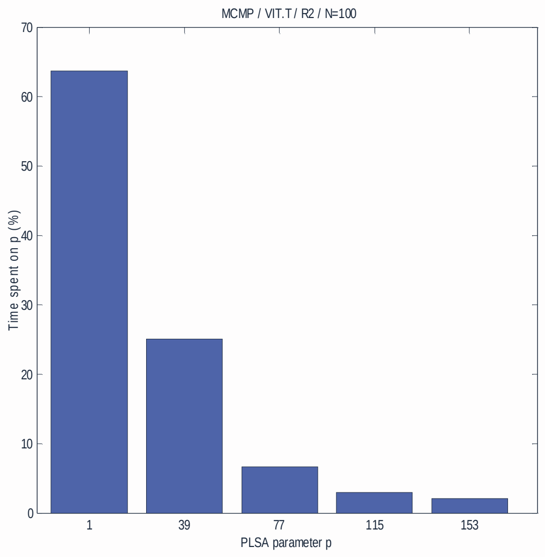


Figure 17. Percent of time spent on each parameter in range R^1 (a) and in range R^2 (b) for VIT.T.

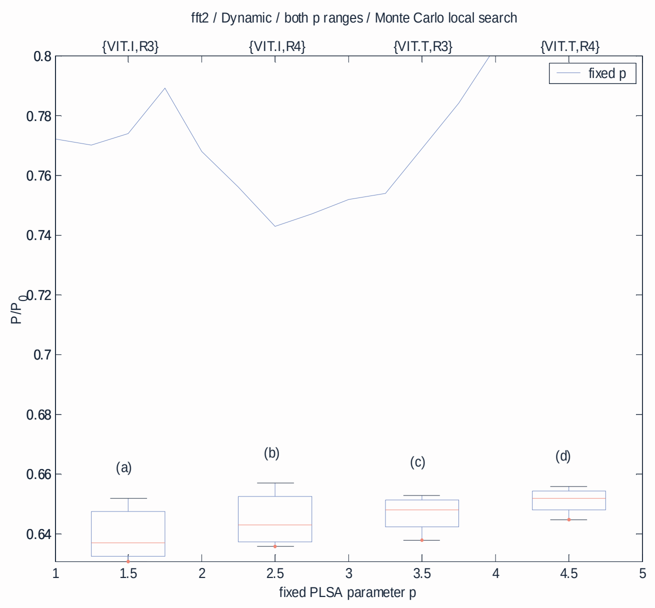


Figure 16. **Dynamic heating** for **voltage scaling** with different parameter ranges depicted by the four box plots—$(a)\{\mathrm{VIT.I}, R^3\}$, $(b)\{\mathrm{VIT.I}, R^4\}$, $(c)\{\mathrm{VIT.T}, R^3\}$, $(d)\{\mathrm{VIT.T}, R^4\}$. VIT.T refers to variable iterations and time per parameter, with the next parameter taken if, for a given time, the solution has not improved. The solid curve depicts results for the standard hybrid approach. All the dynamic schemes outperform the standard hybrid (fixed p) approach, with the lowest average power obtained for (a) VIT.I which utilizes the broader parameter range. The voltage scaling results appear to be less sensitive to the parameter range than the results for the MCMP problem.

5.4.5 Comparison of Heating Schemes

The results indicate that the choice of parameter p does affect the outcome of the optimization process. For the MCMP application, there is a pronounced region for fixed p values around $p = 39$ where the hybrid (with p fixed) performs best. This is illustrated in Figure 11 (also shown as the solid curves in Figures 13 and 15). This is due to the trade-offs in accuracy and complexity with p. For smaller values of p, a larger number of iterations can be performed (cf. Figure 10(e)). It seems that there is a point beyond which increasing p decreases the performance of the hybrid algorithm. As illustrated in Figure 18, continuously increasing p starting from $p = p_{\mathrm{min}}$ also increases the accuracy $A(p)$ of the PLSA and therefore the effectiveness of the overall algorithm. However, when a certain run-time complexity $C(p_{\mathrm{opt}})$ of the PLSA is reached, the benefit of higher accuracy may be outweighed by the disadvantage that the number of iterations that can be explored is smaller. As a consequence, values greater than p_{opt} may reduce the overall performance as the number of iterations is too low. Figure 14(e) depicts the performance of the hybrid with p fixed for the voltage scaling application on the fft2 graph. It can be seen that the region of best performance is not as pronounced as in the MCMP application.

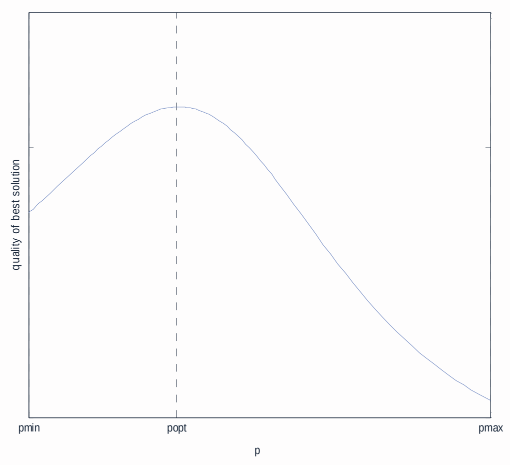


Figure 18. Relationship between the value for p and the outcome of the optimization process.

The observation that certain parameter ranges appear to be more promising than the entire range of permissible p values leads to the question of whether the heating schemes can do better when using the reduced range. One would expect that the static heating schemes, for which the number of iterations at each parameter is fixed beforehand, would benefit the most from the reduced range, since the hybrid would not be ‘forced’ to run beyond p_{opt}. The dynamic heating schemes, by contrast, will continue to operate on a given parameter as long as the quality of the solution is improving. For the MCMP application, range $R^2 = [1, 39, 77, 116, 153]$ is centered around the best fixed p values. For the voltage scaling application, range $R^4 = [2.25, 2.5, 2.75, 3, 3.25]$ is centered around the best fixed p values. Figures 13 through 16 compare the performance over the two parameter ranges. For the static heating optimizations in Figures 13 and 14, the performance is improved by using the reduced parameter ranges. The dynamic heating optimization in Figure 15 shows a smaller relative improvement. The dynamic heating optimization in Figure 16 actually shows a benefit to using the expanded parameter range. It is important to note that in practice one would not know about the characteristics of the different parameter ranges without first performing an optimization at each value. This would take much longer than the simulated heating optimization itself, so in practice the broader parameter range would probably be used. The data for fixed p for the MCMP problem (Figure 11) demonstrate that it can be difficult to find the optimal p value and that this optimum may be isolated, i.e. p values close (e.g. 100) to the optimum yield much worse results. If we calculate the median over all p values tried, the mean performance of the constant p approach is worse than the median performance of the FTS and VIT methods.

Figure 19 compares the results of the different heating schemes for the MCMP application with population size $N = 100, 200, 50$ and parameter range R^1. Figure 20 compares the heating schemes for the voltage scaling

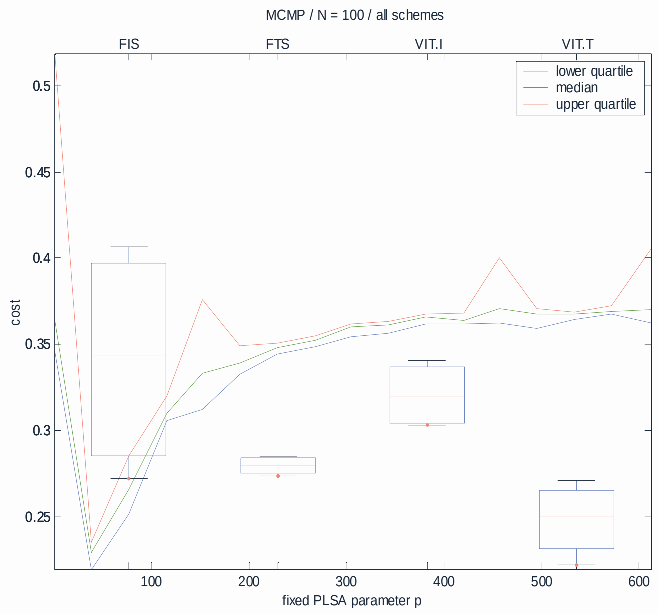
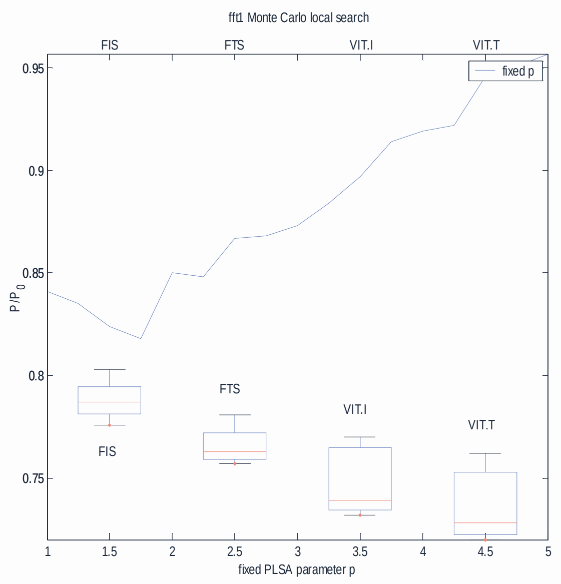


Figure 19. **Comparison of heating schemes for MCMP** with $N = 100$. The two box plots on the left correspond to the static heating schemes. The two box plots on the right correspond to dynamic heating schemes. The best results (lowest memory cost) are obtained for the VIT.T dynamic heating scheme. This refers to variable iterations and time per parameter, where the parameter is incremented if the overall solution does not improve after a pre-determined time, called the stagnation time. The solid curve represents the standard hybrid approach applied at different values of fixed p. The point p=39 slightly outperforms the VIT.T scheme.

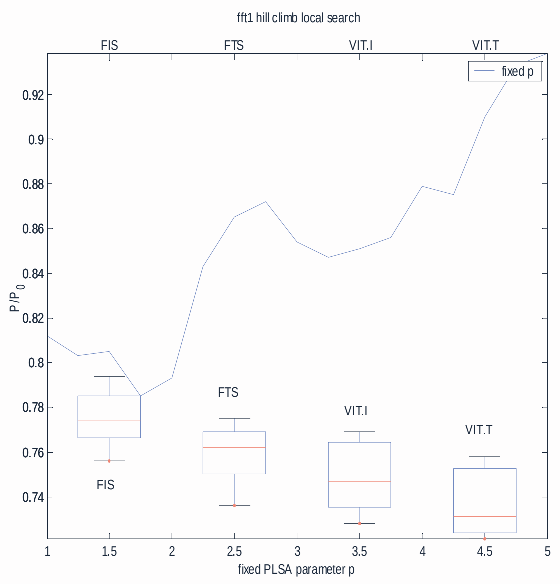
application on different graphs for both types of local search.

Comparing the heating schemes across all different cases, we see that the dynamic heating schemes performed better in general than the static heating schemes. For all cases, the best heating scheme was dynamic. For the binary knapsack problem and the voltage scaling problem, simulated heating always outperformed the standard hybrid approach.

For the MCMP problem, there was one PLSA parameter where the standard hybrid approach slightly outperformed the dynamic, simulated heating approach. We note that in practice, one would need to scan the entire range of parameters to find this optimal value of fixed p, which is in fact equivalent to allotting much more time to this method. Thus, we can say that the simulated heating approach outperformed the standard hybrid approach in the cases we studied.

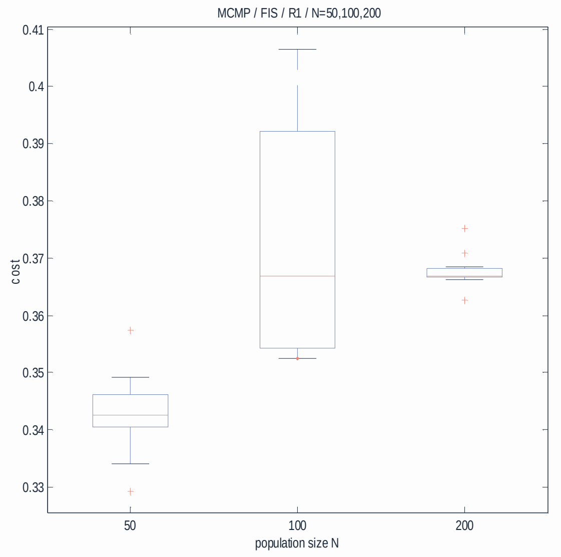


(a)

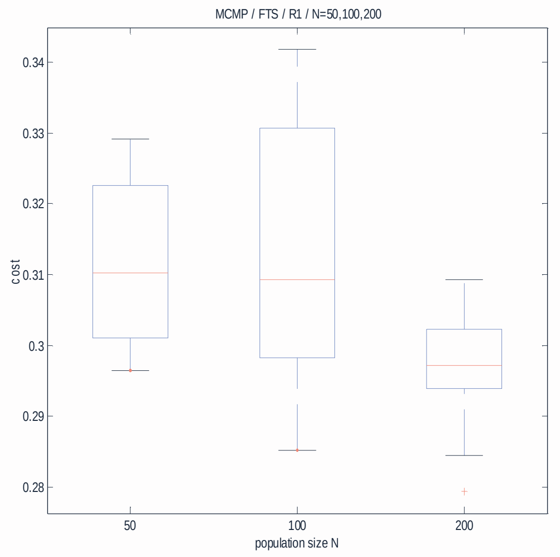


(b)

Figure 20. **Comparison of heating schemes for voltage scaling** with (a) Monte Carlo and (b) hill climb local search. The 2 box plots on the left correspond to the FIS and FTS static heating schemes, while the box plots on the right correspond to dynamic heating schemes VIT.I and VIT.T. The line across the middle of the boxes represents the median over the runs, while the 'whisker lines' are drawn at the 10th and 90th percentiles. The solid curve represents the standard hybrid approach using fixed p at various values of p. In this application, all the simulated heated schemes outperformed the standard hybrid approach. The best results were obtained for the dynamic VIT.T scheme.



(a)



(b)

Figure 21. Static heating with different population sizes—(a) FIS, (b) FTS.

5.4.6 Effect of Population Size

Figure 21 shows the effect of the population size for MCMP for the static heating schemes. Figure 22 shows the effect of population size on the dynamic heating schemes for MCMP.

For FIS, smaller population sizes seem to be preferable. The larger number of iterations that can be explored for $N = 50$ may be an explanation for the better performance. In contrast, the heating scheme FTS achieves better results when a larger population $N = 200$ is used. For the dynamic heating schemes, the results seem to be less sensitive to the population size.

5.4.7 Discussion

We summarize several trends in our experimental data.

- The dynamic variants of the simulated heating technique outperformed the standard hybrid global/local search technique.
- When employing the standard hybrid method utilizing a fixed parameter p, an optimal value of p may be isolated and difficult to find in advance.
- Such optimal values of p depend on the application.
- When performing simulated heating, our experiments show that choosing the parameter range to lie around the best fixed p values yields better results than using the broadest range in most cases. However, using the broader range still produces good results, and this is the method most likely to be used in practice.
- The dynamic heating schemes show less sensitivity to this parameter range.
- Overall, the dynamic heating schemes performed better than the static heating schemes.

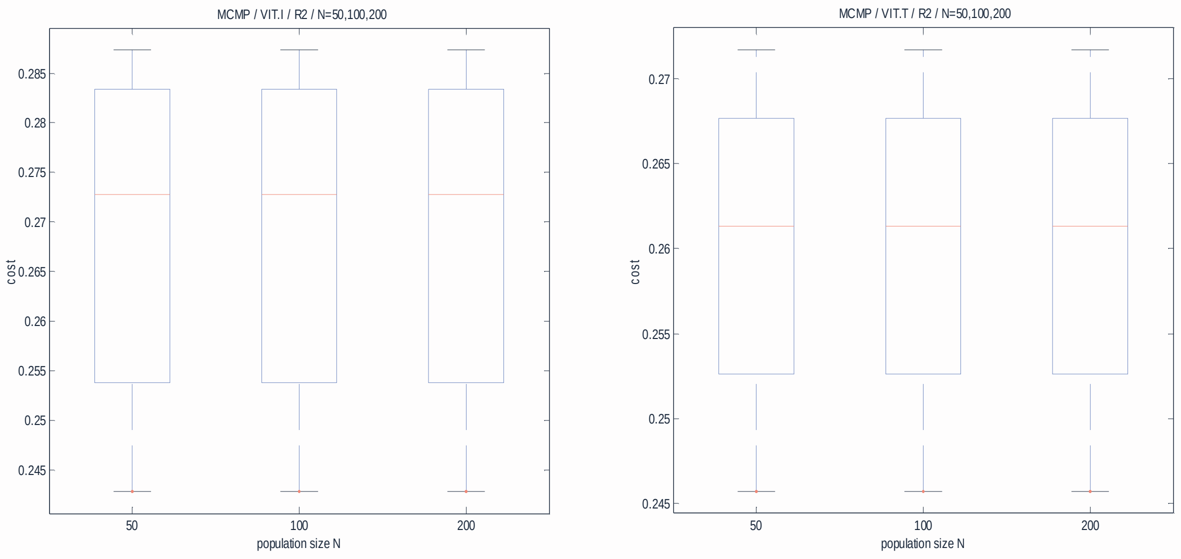


Figure 22. Dynamic heating with different population sizes—(a) VIT.I, (b) VIT.T.

- The dynamic heating schemes were also less sensitive to the population size of the global search algorithm.

6 Conclusions

Efficient local search algorithms, which refine arbitrary points in a search space into better solutions, exist in many practical contexts. In many cases, these local search algorithms can be parameterized so as to trade off time or space complexity for optimization accuracy. We call these parameterized local search algorithms (PLSAs). We have shown that a hybrid PLSA/EA (parameterized local search/evolutionary algorithm) can be very effective for solving complex optimization problems. We have demonstrated the importance of carefully managing the run-time/accuracy trade-offs associated with EA/PLSA hybrid algorithms, and have introduced a novel framework of simulated heating for this purpose. We have developed both static and dynamic trade-off management strategies for our simulated heating framework, and have evaluated these techniques on the binary knapsack problem and two complex, practical optimization problems with very different structure. These problems have vast solution spaces, and underlying PLSAs that exhibit a wide range of accuracy/complexity trade-offs. We have shown that, in the context of a fixed optimization time budget, simulated heating better utilizes the time resources and outperforms the standard fixed parameter hybrid methods. In addition, we have shown that the simulated heating method is less sensitive to the parameter settings.

7 Appendix

7.1 Implementation Details for MCMP

To solve the MCMP we use a GSA/PLSA hybrid as discussed in section 3 where an evolutionary algorithm is the GSA and CDPPO is the PLSA. The evolutionary algorithm and parameterized CDPPO are explained below.

GSA: Evolutionary Algorithm for MCMP

Each solution s is encoded by an integer vector, which represents the corresponding schedule, i.e., the order of actor executions (*firings*). The decoding process that takes place in the local search/evaluation phase (step 5 of Definition 1) is as follows:

- first a repair procedure is invoked, which transforms the encoded actor firing sequence into a valid flat schedule,

- then the parameterized CDPPO is applied to the resulting flat schedule in order to compute a (sub)optimal looping, and afterwards the data requirement (buffering cost) $D(s)$ and the program requirement (code size cost) $P(s)$ of the software implementation represented by the looped schedule are calculated based on a certain processor model.

Finally, both $D(s)$ and $P(s)$ are normalized (the minimum values $D_{\min}$ and $P_{\min}$ and maximum values $D_{\max}$ and $P_{\max}$ for the distinct objectives can be determined beforehand) and a fitness is assigned to the solution s according to the following formula:

$$F(s) = 0.5\frac{D(s) - D_{\min}}{D_{\max} - D_{\min}} + 0.5\frac{P(s) - P_{\min}}{P_{\max} - P_{\min}}. \quad (10)$$

Note that the fitness values are to be minimized here.

PLSA: Parameterized CDPPO for MCMP

The “unparameterized” CDPPO algorithm was first proposed in [4]. CDPPO computes an optimal parenthesization in a bottom-up fashion, which is analogous to dynamic programming techniques for matrix-chain multiplication [9]. Given a dataflow graph $G = (V, E)$ and an actor invocation sequence (flat schedule) $f_1, f_2, \ldots, f_n$, where each $f_i \in V$, CDPPO first examines all 2-invocation *sub-chains* $(f_1, f_2), (f_2, f_3), \ldots, (f_{n-1}, f_n)$ to determine an optimally-compact looping structure (*subschedule*) for each of these sub-chains. For a 2-invocation sub-chain (f_i, f_{i+1}), the most compact subschedule is easily determined: if $f_i = f_{i+1}$, then $(2f_i)$ is the most compact subschedule, otherwise the original (unmodified) subschedule $f_i f_{i+1}$ is the most compact. After the optimal 2-node subschedules are computed in this manner, these subschedules are used to determine optimal 3-node subschedules (optimal looping structures for subschedules of the form f_i, f_{i+1}, f_{i+2}); and the 2- and 3-node subschedules are then used to determine optimal 4-node subschedules, and so on until the *n*-node optimal subschedule is computed, which gives a minimum code size implementation of the input invocation sequence $f_1, f_2, \ldots, f_n$.

Due to its high complexity, CDPPO can require significant computational resources for a single application—e.g., we have commonly observed run-times on the order of 30-40 seconds for practical applications. In the context of global search techniques, such performance can greatly limit the number of neighborhoods (flat schedules) in the search space that are sampled. To address this limitation, however, a simple and effective parameterization emerges: we simply set a threshold M on the maximum sub-chain (subschedule) size to which optimization is attempted. This threshold becomes the parameter of the resulting *parameterized CDPPO* (PCDPPO) algorithm.

In summary, PCDPPO is a parameterized adaptation of CDPPO for addressing the schedule looping problem. The run-time and accuracy of PCDPPO are both monotonically nondecreasing functions of the algorithm “threshold” parameter *M*. In the context of the memory minimization problem, PCDPPO is a genuine PLSA.

7.2 Voltage Scaling Implementation

To solve the dynamic voltage scaling optimization problem we use a GSA/PLSA hybrid as discussed in section 3 where an evolutionary algorithm is the GSA and the PLSA is either a hill climbing or Monte Carlo search utilizing the period graph. Two different local search strategies were implemented—hill climbing [22] and Monte Carlo [19]. Pseudo-code for both local search methods is shown in Figures 23 and 24. The benefit of using a local search algorithm is that within a restricted voltage range we can use the period graph estimator for the throughput, which is much faster than performing a simulation. The local search algorithms are explained further below.

GSA: Evolutionary Algorithm for Voltage Scaling

Each solution s is encoded by a vector of positive real numbers of size N representing the voltage assigned to each of the N tasks in the application. The one-point crossover operator randomly selects a crossover point within a vector then interchanges the two parent vectors at this point to produce two new offspring. The mutation operator randomly changes one of the elements of the vectors to a new (positive) value. At each generation of the EA an entirely new population is created based on the crossover and mutation operators. The crossover probability was 0.9, the mutation probability was 0.1, and the population size was 50.

Voltage Scaling PLSA1: Hill Climb Local Search

For the hill climbing algorithm, we defined a parameter δ, which is the voltage step, and a resimulation threshold r, which is the maximum amount that the voltage vector can vary from the point at which the period graph was calculated. We ran the algorithm for I iterations. So for this case, the PLSA L had 3 parameters I, r, and δ. One iteration of local search consisted of changing the node voltages, one at a time, by $\pm\delta$, and choosing the direction in which the objective function was minimized. From this, the worst case cost $C(I, r, \delta)$ for I iterations would correspond to evaluating the Objective function $3I$ times, and resimulating $(I/\lceil r/\delta \rceil)$ times. For our experiments we fixed I and δ and defined the local search parameter as $p = 1/r$. Then for smaller p (corresponding to larger resimulation threshold) the voltage vector can move a greater distance before a new simulation is required. For a fixed number of iterations I in the local search, a smaller p corresponds to a shorter running time $C(p)$ for $L(p)$.

```
HillClimb()
Input: voltage vector $V_{in}$ of size N, number of iterations I
Period graph $G$ with N tasks for which the execution time of task i is scaled by $V[i]$.
$V_{obj}$ is the objective function derived from maximum cycle mean of $G$ scaled by $V$
$V_{resim}$ is the resimulation threshold distance
Output: voltage vector $V_{out}$ and new objective score

$\mathrm{LowScore} \leftarrow \infty$
$V \leftarrow V_{in}$
for (k = 0; k < I; k++) {
    for (i = 0; i < N; i++) {
        $V_0 \leftarrow V[i]$
        $V[i] \leftarrow V_0(1+\delta)$
        $f_1 = V_{obj}(V[i], G)$
        $V[i] = V_0(1-\delta)$
        $f_2 = V_{obj}(V[i], G)$
        $V[i] = V_0$
        $f = V_{obj}(V[i], G)$
        if( $f_1 < f$ )
            $V[i] = V_0(1+\delta)$
        else if($f_2 < f$)
            $V[i] = V_0(1+\delta)$
        Let $D$ be $\|V[i] - V_{in}\|$
        if($D < V_{resim}$)
            Resimulate and rebuild G
            $V_{in} = V$
    }
    $\mathrm{score} = V_{obj}(V[i], G)$
    if($\mathrm{score} < \mathrm{LowScore}$)
        $\mathrm{LowScore} = \mathrm{score}$
        $V_{out} = V$
}
```

Figure 23. Pseudo-code for hill climb local search for voltage scaling.

The accuracy $A(p)$ is lower, since the accuracy of the period graph estimate decreases as the voltage vector moves farther away from the simulation point.

Voltage Scaling PLSA2: Monte Carlo Local Search

In the Monte Carlo algorithm, we generated N random voltage vectors within a distance D from the input vector. For all points within a resimulation threshold r, we used the period graph to estimate performance. A greedy strategy was used to evaluate the remaining points. Specifically, we selected one of the remaining points at random, performed a simulation to construct a new period graph, and used the resulting estimator to evaluate all points within a distance r from this point. If there were points remaining after this, we chose one of these and repeated the process.

```
Monte Carlo()
Input: voltage vector $V_{\text{in}}$ of size N
Number $R$ of random vectors to generate
Period graph $G$ with N tasks for which the execution time of task i is scaled by $V[i]$ .
$V_{\text{obj}}$ is the objective function derived from maximum cycle mean of $G$ scaled by $V$
$K$ is the input constraint on $V_{\text{obj}}$ .
$V_{\text{resim}}$ is the resimulation threshold distance
Output: voltage vector $V_{\text{out}}$ and new objective score

Generate a list $L_{\text{rand}}$ of $R$ random vectors uniformly distributed with a distance of no
more than $V_{\text{resim}}$ from $V_{\text{in}}$ .

Evaluate $q = \sum_i (V[i])^2$ for each vector in $L_{\text{rand}}$ .

Sort $L_{\text{rand}}$ according to lowest $q$ first.

for(j=0; j < R; j++){
    Pop head of list $L_{\text{rand}}$ to get $V$
    Scale $G$ by $V$
    $f = V_{\text{obj}}(V[i], G)$

    if ( $f < K$ ){
        return( $V_{\text{out}} = V$ )
    }
}
```

Figure 24. Pseudo-code for Monte Carlo local search for voltage scaling.

For the experiments we fixed N and D and defined local search parameter $p = 1/r$. As for the hill climbing local search, smaller values of p correspond to shorter run-times and less accuracy for the Monte Carlo local search.

7.3 Methodology

Simulated heating applied to the memory cost minimization application was evaluated on a complex, real-life *CD2DAT* application benchmark. This benchmark performs a sample-rate conversion from a compact disk (CD) player output (44.1 kHz) to a digital audio tape (DAT) input (48 kHz). Due to the extensive multirate processing involved, this application requires over 600 actor invocations per flat schedule, and consequently, the quartic complexity of the original (full-strength) CDPPO becomes heavily cumbersome for probabilistic global search techniques. The CD2DAT application has been used extensively as a benchmark to evaluate dataflow-based software optimization techniques [5,6].

For the voltage scaling optimization we utilized 6 common DSP benchmarks. For each benchmark, we set the voltages to a nominal 5 volt value and calculated the system throughput. This throughput was then used as a constraint while the optimization attempted to minimize the average power of the system.

8 References

[1] E. Balas and E. Zemel, “An Algorithm for Large Zero-One Knapsack Problems,” *Operations Research*, vol. 28, pages 1130-1154, 1980.

[2] N. K. Bambha and S. S. Bhattacharyya, “A Joint Power/Performance Optimization Technique for Multiprocessor Systems Using a Period Graph Construct,” *Proceedings of the International Symposium on Systems Synthesis*, pages 91-97, September 2000.

[3] N. K. Bambha and S. S. Bhattacharyya, “A Period Graph Throughput Estimator for Multiprocessor Systems,” Technical Report UMIACS-TR-2000-49, Institute for Advanced Computer Studies, University of Maryland at College Park, July 2000.

[4] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, “Optimal Parenthesization of Lexical Orderings for DSP Block Diagrams,” *Proceedings of the International Workshop on VLSI Signal Processing*, IEEE press, October 1995. Sakai, Osaka, Japan.

[5] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*, Kluwer, Norwell, MA, 1996.

[6] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, “Synthesis of Embedded Software from Synchronous Dataflow Specifications,” *Journal of VLSI Signal Processing Systems*, vol. 21, no. 2, pages 151-166, June 1999.

[7] T. Burd and R. Broderson, “Design Issues for Dynamic Voltage Scaling,” In *Proceedings of 2000 International Symposium on Low Power Electronics and Design*, pages 76-81, July 2000.

[8] A. P. Chandrakasan, S. Sheng, and R. W. Broderson. “Low-power CMOS digital design,” *IEEE Journal of Solid-State Circuits*, 27(4):473—484, 1992.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1992.

[10] A. Dasdan and R. K. Gupta, “Faster Maximum and Minimum Mean Cycle Algorithms for System-Performance Analysis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):889-899, October 1998.

[11] L. Davis, **Handbook of Genetic Algorithms**, Van Nostrand Reinhold, New York, 1991.

[12] T. Feo and M. Resende, “A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem,” *Operations Research Letters*, vol. 8, pages 67-71, 1989.

[13] T. Feo, K. Venkatraman, and J. Bard, “A GRASP for a Difficult Single Machine Scheduling Problem,” *Computers and Operations Research*, vol. 18, pages 635-643, 1991.

[14] C. Fleurent and J. Ferland, “Genetic Hybrids for the Quadratic Assignment Problem,” DIMACS Series in Discrete

Mathematics and Theoretical Computer Science, vol. 16, 1994.

[15] D. Goldberg, **Genetic Algorithms in Search, Optimization, and Machine Learning**, Addison Wesley, 1989.

[16] D. E. Goldberg and S. Voessner, “Optimizing Global-Local Search Hybrids,” *GECCO’99*, volume 1, pages 220-228, San Francisco, CA, 1999. Morgan Kaufmann.

[17] H. He, J. Xu, and X. Yao, “Solving Equations by Hybrid Evolutionary Computation Techniques,” IEEE Transactions on Evolutionary Computation, vol. 4, no. 3, pages 295-304, September 2000.

[18] H. Ishibuchi and T. Murata, “Multi-Objective Genetic Local Search Algorithm, *IEEE Conference on Evolutionary Computation (ICEC ‘96)*, pages 119-124, 1996.

[19] M. Kalos and P. Whitlock, **Monte Carlo Methods**, Wiley-Interscience, New York, 1986.

[20] S. Kazarlis, S. Papadakis, J. Theocharis, “Microgenetic Algorithms as Generalized Hill-Climbing Operators for GA Optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 3, June 2001, pages 204-217.

[21] B. W. Kernighan and S. Lin, “An Efficient Heuristic Procedure for Partitioning Graphs,” *Bell System Technical Journal*, vol. 49, pages 291-307, 1970.

[22] D. Kreher, D. Stinson, **Combinatorial Algorithms: Generation, Enumeration, and Search,** CRC Press, pages 157-158, 1999.

[23] E. Lawler, **Combinatorial Optimization: Networks and Matroids.** New York: Holt, Rhinehart and Winston, 1976.

[24] E. A. Lee and S. Ha, “Scheduling Strategies for Multiprocessor Real Time DSP,” *Proceedings of the Global Telecommunications Conference*, November 1989.

[25] E. A. Lee and D.G. Messerschmitt, “Synchronous Dataflow,” Proceedings of the IEEE, vol. 75, no. 9, pages 1235-1245, September 1987.

[26] D. Marculescu, “On the Use of Microarchitecture-Driven Dynamic Voltage Scaling”, *Proceedings of ISCA* 2000.

[27] P. Merz and B. Freisleben, “Genetic Algorithms for Binary Quadratic Programming, *GECCO ‘99*, volume 1, pages 417-424, San Francisco, CA, 1999, Morgan Kaufmann.

[28] P. Merz and B. Freisleben, “A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem,” *International Congress on Evolutionary Computation (CEC’99)*, IEEE Press, pages 2063-2070, 1999.

[29] T. Pering, T. Burd, and R. Broderson, “The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms,” *Proceedings of International Symposium on Low Power Electronics and Design*, pages 76-81, August 1998.

[30] D. Pisinger, “Core problems in Knapsack Algorithms,” Technical Report 94/26, DIKU, University of Copenhagen, Denmark, June 1994.

[31] D. Pisinger, “An Expanding-Core algorithm for the Exact 0-1 Knapsack Problem,” *European Journal of Opeations Research*, vol. 87, pg. 175-177, 1995.

[32] S. Reiter and G. Sherman, “Discrete Optimizing,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 13, pages 864-889, 1965.

[33] M. Ryan, J. Debuse, G. Smith, and I. Whittley, “A Hybrid Genetic Algorithm for the Fixed Channel Assignment

Problem," *GECCO'99*, volume 2, pages 1707-1714, San Francisco, CA 1999. Morgan Kaufmann.

[34] G. C. Sih and E. A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Transactions on Parallel and Distributed Systems*, 4(2):75-87, February 1993.

[35] S. Sriram, and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., 2000.

[36] M. Vazquez and D. Whitley, "A Hybrid Genetic Algorithm for the Quadratic Assignment Problem," GECCO 2000.

[37] E. Zitzler, J. Teich, and S. S. Bhattacharyya, "Optimizing the Efficiency of Parameterized Local Search Within Global Search: A Preliminary Study," *Proceedings of the Congress on Evolutionary Computation*, pages 365-372, San Diego, California, July 2000.

[38] E. Zitzler, J. Teich, and S. S. Bhattacharyya, "Evolutionary Algorithm Based Exploration of Software Schedules for Digital Signal Processors," *GECCO '99*, volume 2, pages 1762-1769, San Francisco, CA, 1999. Morgan Kaufmann.

[39] E. Zitzler, J. Teich, and S. S. Bhattacharyya, "Multidimensional Exploration of Software Implementations for DSP Algorithms," *Journal of VLSI Signal Processing*, vol. 24, no. 1, pages 83-98, February 2000.

[40] http://www.diku.dk/~pisinger/codes.html