Sustainable production automation - Energy optimization of robot cells

Sustainable Production Automation - Energy Optimization of Robot Cells

Oskar Wigström and Bengt Lennartson

***Abstract*— This paper concerns the reduction of energy use in manufacturing industry. If individual robot movements in a system are preprocessed using Dynamic Programming, one can attain a Mixed Integer Nonlinear Program (MINLP) which models the energy consumption of the complete system. This model can then be solved to optimality using mathematical programming. We have previously shown proof of concept for this energy reduction method. In this paper, we apply state of the art MINLP methods to a number of problems in order benchmark their effectiveness. Algorithms used are Nonlinear Programming based Branch and Bound (NLP-BB), Outer Approximation (OA), LP/NLP based Branch and Bound (LP/NLP-BB) and Extended Cutting Plane (ECP). Benchmarks show that the NLP-BB does not perform well for nonlinear scheduling problems. This is due to the weak lower bounds of the integer relaxations. For scheduling problems with nonlinear costs, ECP and in particular LP/NLP-BB are shown to outperform both NLP-BB and OA. The resulting energy optimal schedules for the examples show a significant decrease in energy consumption.**

I. Introduction

Minimizing energy consumption in industrial applications is important both from an environmental and economical point of view. One of several approaches to this problem is to improve existing hardware solutions. Energy optimization of mechatronic devices is well investigated in [1], [2], [3]. Minimizing the energy cost for trajectories in robot applications is in itself a big research field, see e.g. [4], [5]. From a system design perspective, a selection and matching of efficient design solutions for pre-defined operations is studied in [6], [7]. Also, two approaches where idle time between the operations is used to reduce velocities and accelerations is presented in [8]. This is however without concern to the energy consumption.

In a typical manufacturing system, multiple robots will often work together in close proximity each other, e.g. as in Figure 1. One way to avoid collisions between robots is to require booking of any areas which might be occupied by more than one robot. Previously, we have presented a method for the energy optimization of systems where multiple moving devices share common resources [9]. The method consists of two steps, one which preprocesses possible movements within the system, and another which schedules these processed movements. Where our previous paper focused mainly on the preprocessing step and showed as a proof of concept how the scheduling could be performed, this paper delves deeper into the scheduling part.

It is reasonable to assume that the spatial paths of moving devices in a production system are known, or can at least be fixed. This is in particular true for industrial robots. For example, a process designer would like a robot endpoint to perform a linear translation. Then with a fixed path for the endpoint, the evolution of the joint coordinates are known by inverse kinematics. Also, the spatial paths are most often determined before scheduling is applied. This is because to avoid collisions in robotic systems, the areas where there is risk for collision must be identified before scheduling. The scheduling problem is thus a matter of driving the robots along their paths such that mutual exclusion is fulfilled for common resources and other constraints are upheld.

The previously mentioned preprocessing step consists of using Dynamic Programming (DP) to generate the optimal cost for each path (movement) as a function of time. These functions take the form of polynomials and are then used to form the total energy consumption for the entire system. This is then used as a cost function in the global scheduling problem, which takes the form of a convex Mixed Integer Nonlinear Program (MINLP). In [9], a small case study was presented but the solution method consisted of simply using explicit enumeration of the feasible integer solutions and using a Nonlinear Programming (NLP) solver to solve each explicit instance. In this paper however, we apply state of the art MINLP methods to study the efficiency of different solution methods, as well as show the tractability of our method for problems of various characteristics.

The paper is structured as follows. Section II contains

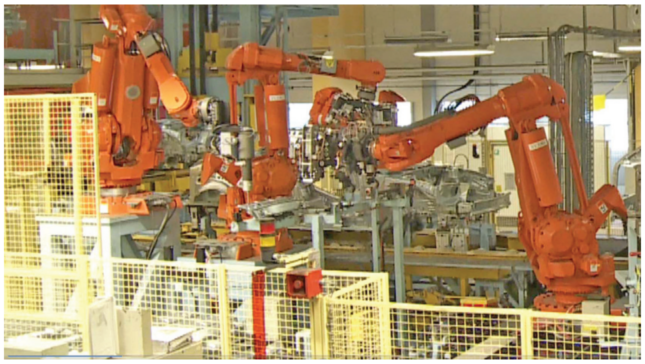
Fig. 1. Three robots working in close proximity to each other.

This work was carried out at the Wingquist Laboratory VINN Excellence Center within the Area of Advance – Production at Chalmers, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) and the Swedish Research Council. The support is gratefully acknowledged.

O. Wigström, B. Lennartson, Automation Research Group, Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden, oskar.wigstrom@chalmers.se

introduction to convex MINLP and introduces the algorithms used for benchmarking in this paper. Section III recapitulates the DP method used to generate the cost function for the scheduling problem. Section IV presents the scheduling model and Section V the case study. Finally Section VI contains a brief discussion.

II. Convex MINLP Methods

The constraints arising from a scheduling problem are that of mixed logical linear. To describe execution and simple temporal ordering of operations, it is enough with linear equalities and inequalities. But when two operations are subject to the mutual exclusion of a resource, one of the two operations must occur before the other. The resulting mathematical description consists of two linear constraints with an 'exclusive or' (XOR) statement describing that only one of the two linear constraints should hold. Luckily, logic expressions can be encoded into mixed integer linear constraints [10]. Scheduling problems can thus be modeled and solved using mixed integer methods.

In this paper, we are concerned with minimizing the combined energy consumption of a number of robots working together. In [11] we showed how to generate the minimum cost of a nonlinear time invariant criteria as a function of execution time. A quick summary is provided in Section III. These energy functions are approximated as convex polynomials and form the cost function. Because the energy cost is convex nonlinear in general, the complete scheduling problem is that of a convex MINLP. In an algebraic form, a MINLP is expressed as

$$\begin{aligned} \underset{x,y}{\text{minimize}} \quad & f(x,y) \\ \text{subject to} \quad & g(x,y) \leq 0 \\ & x \in X \\ & y \in Y \quad \text{integer}, \end{aligned}$$

where $f : \mathbb{R}^{n+m} \to \mathbb{R}$ is a possibly nonlinear objective function and $g : \mathbb{R}^{n+m} \to \mathbb{R}^k$ is a possibly nonlinear constraint function. In our case, we require both f and g to be convex. From here on we will refer to convex MINLP as just MINLP. The two sets of decision variables are x and y, where y is required to be integer valued.

An excellent and substantial review which summarizes the current state of MINLP can be found in [12]. There are a number of different methods and packages that implement these methods. In this paper, we have used the open source solver Bonmin [13]. We chose to benchmark four popular choices of algorithms: Nonlinear Programming based Branch and Bound (NLP-BB), Outer Approximation (OA), LP/NLP based Branch and Bound (LP/NLP-BB) and Extended Cutting Plane (ECP). The following provides a short summary of the algorithms.

(NLP-BB) The theory used for the branch and bound algorithms used to solve MILP problems, first presented in [14], do not have any requirement on linearity [15]. A nonlinear implementation, proposed in [16], works similar to its linear counterpart. A tree of problems, yet to be processed is defined. The tree is initialized with an integer relaxed problem, an NLP. The solution to this relaxed NLP provides a lower bound on the objective. Iteratively, integer variables are branched upon and constrained, such that NLP subproblems, with some integers fixed and other relaxed, are created. The resulting optimal solution for each sub problem provides a lower bound for the current branch. When all integer variables have been constrained such that they form an assignment, the solution provides an upper bound for the MINLP. Once an upper bound has been found, any nodes with a greater lower bound can be removed from the tree of problems. The algorithm terminates when all branches of the tree have been explored or removed.

(OA) First presented in [17], OA utilizes the fact that a MINLP is equivalent to a MILP of finite size, see also [18]. A MILP formulation, or reduced Master Problem (MP), which yields the same optimal solution as the MINLP can be constructed by linearizations of the MINLP. Usually, the algorithm is initiated by solving the integer relaxed MINLP, the MINLP is then linearized at the solution point and the resulting constraints are added to the MP. Solving the MP yields an integer solution, which is then used to form an NLP. Solving the NLP provides yet another point at which the MINLP is linearized. This procedure is repeated iteratively. Note that the MP does not necessarily have to be solved to optimality, as any valid integer assignment which yields a better solution than the current upper bound can be used in the NLP.

(LP/NLP-BB) First presented in [19], LP/NLP-BB functions similarly to OA. Instead of solving a sequence of MPs, a branch and bound tree with LP-relaxations is started on the initial MP. The algorithm progresses much like a branch and bound MILP solver would. But each time a partial integer solution is found, it can be used to generate an NLP. Different rulesets exist which define how often an NLP should be generated. The resulting NLP solution point is linearized and used to update the tree of LPs. The LP/NLP-BB thus differs to OA in the regard that the MILP solver need not restart. See [13] for details on Bonmin's implementation.

(ECP) Introduced in [20], this method requires no NLP solver to function. The algorithm is based on iteratively solving a master MILP problem. At each iteration, the most violated constraints of the solution are linearized and added to the master problem. The algorithm terminates as the maximum constraint violation is smaller than some prespecified tolerance.

Note that in general, for each integer relaxed MINLP that is solved, one would like the convex hull of the relaxation to be as tight as possible. While there exists techniques which can be used to generate tight convex hull relaxations, this will be of little or no help for our problem. As pointed out in [21], even the tightest convex hull which arises from relaxations of scheduling disjunctions is so large that it is most likely useless in practice.

III. Trajectory Planning

A trajectory planning problem entails finding the control signal which will move a manipulator or other moving device along a predefined geometric path, while upholding its dynamical constraints. A literature review covering the last three decades can be found in [22]. In this paper, we use the dynamic programming method described in [11]. The grid dimensionality is two and is unchanged for an increasing number of spatial dimensions. A favourable property of this particular method is that one execution will yield the optimal cost for the entire spectrum of end times.

The following is a truncated formulation, for more detail see [11]. Let the known spatial path be defined by a function $x_p(\tau)$, a parametrized curve dependent on one single variable $\tau(t)$. The time optimal trajectory can be used to define x_p and its derivatives. This implies that τ is the time scale for the time optimal trajectory, x_p. For example, defining $\tau = t$ would result in the time optimal trajectory. The relationship between x and x_p can therefore be expressed as

$$x(t) = x_p(\tau(t)), \qquad 0 \leq \tau \leq \tau_f, \tag{1}$$

where $\tau(t)$ is a monotonically increasing function with a starting value of 0 and final value τ_f, where τ_f in our case corresponds to the time optimal execution time. If $\tau(t_f) = \tau_f$, then t_f is the new final execution time of the dynamically scaled trajectory. Differentiating (1) with regard to time yields expressions for speed and acceleration. Note that we will restrict ourselves to considering the second derivative of x. Given that x and its derivatives are functions of τ and $x_p(\tau)$, and as the latter is known, any cost function on the form

$$c(t_f) = \int_0^{t_f} g(x, \dot{x}, \ddot{x})\ dt, \tag{2}$$

can be rewritten as

$$c(t_f) = \int_0^{t_f} g(\dot{\tau}(0), \ddot{\tau})\ dt, \tag{3}$$

where t_f is the final time of the trajectory. The trajectory planning problem is thus a matter of minimizing c with respect to $\dot{\tau}(0)$ and $\ddot{\tau}$.

Let us model τ as a discrete time double integrator with time-varying sampling time h_k that affects the time updates as $t_{k+1} = t_k + h_k$. With an input variable u_k which is constant during the sampling intervals, the discrete state space model for τ is now

$$\begin{bmatrix} \tau_{k+1} \\ \nu_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h_k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tau_k \\ \nu_k \end{bmatrix} + \begin{bmatrix} h_k^2/2 \\ h_k \end{bmatrix} u(t_k), \tag{4}$$

where $\nu = \dot{\tau}$ and for simplicity, we introduce $\tau_{k+1} = \tau(t_{k+1})$, $\tau_k = \tau(t_k)$ and $\nu_{k+1} = \nu(t_{k+1})$, $\nu_k = \nu(t_k)$. The minimization of (3), subject to this discrete time model of the time function $\tau(t)$ can be solved with DP.

For computational reasons, it is convenient to reformulate the problem. Since τ is monotonically increasing it is possible, instead of taking steps along the t-axis in each iteration, to take steps along the τ-axis and let $t_{k+1} = t_k + h_k$ act as a discrete state equation. Let the size of the steps along τ during each iteration be defined by Δ_k, a user defined sampling period or gridding of τ. The reformulated discrete state space is now

$$\begin{bmatrix} t_{k+1} \\ \nu_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} t_k \\ \nu_k \end{bmatrix} + \begin{bmatrix} h_k \\ 2\Delta_k/h_k \end{bmatrix}, \tag{5}$$

which can be more efficiently solved by DP. For details on the reformulation and its computational advantages, please see [9]. From here, DP can be applied to solve the discrete time optimal control problem. The resulting grid with optimal costs can then be approximated as a polynomial. This is done by using standard least squares curve fitting.

IV. MINLP model

Modeling scheduling problems with linear costs can be done in a number of ways. The start and end times of operations can be treated as integers [23] or time can be given implicitly by the order of operations [24]. As a third option, the planning period could be discretized and an operation starting in a specific time instance is modeled by a boolean variable [25]. Since the energy cost is modeled as a convex nonlinear function of execution time, we require a formulation which uses real valued variables for the start and end times of operations. We will therefore adapt the modeling formalism in [23]. Although the original formulation uses integer decision variables to model execution, real valued variables can be used just as well. Recent formulations for the flexible job shop scheduling problem based on the formalism in [23] can be found in for example [26], [27].

In this paper we examine the size of industrial examples which are tractable using our method. Booking resources during multiple operations is concidered in the case study and multiple bookings per operation is supported by the model. However, alternatives and multi-capacity resources are excluded due to space limitations, please refer to [28] for an extended modeling format which supports this.

We formulate the scheduling problem as follows. A problem has a set $\mathcal{J}$ jobs and a set $\mathcal{R}$ resources. Each job, $J \in \mathcal{J}$, consists of a set of operations $\mathcal{O}_J$, we denote each element $O_{J,i}$, $i = [1, ..., h_J]$, where h_J is the number of operations in J. An operation consists of the following parameters:

$O_{J,i}^{min}$ - minimum execution time (constant)
$O_{J,i}^{max}$ - maximum execution time (constant)
$O_{J,i}^{s}$ - start time (decision variable)
$O_{J,i}^{f}$ - final time (decision variable)
$O_{J,i}^{P,k}$ - cost polynomial coefficients, $k = [p_{min}, ..., p_{max}]$

As is, these operations can be performed in any order, but there is also a set of temporal orderings $\mathcal{T}$ which can constrict one operation to be executed after another. We define a temporal ordering $T \in \mathcal{T}$ by the following properties:

T^L - left hand side (pointer to decision variable)
T^R - right hand side (pointer to decision variable)
T_{type} - temporal ordering type $\{eq, leq\}$

We also define, for each resource $R \in \mathcal{R}$, a set of allocation/deallocation pairs $\mathcal{P}_R$ which will govern how operations use resources. Each pair $P_{R,l} \in \mathcal{P}_R$, $l = [1, ..., h_R]$, where h_R is the number of pairs in $\mathcal{P}_R$, describes the allocation and deallocation of a resource.

$P^a_{R,l}$ - allocation (pointer to decision variable)
$P^d_{R,l}$ - deallocation (pointer to decision variable)

The complete scheduling problem is thus given by the tuple

$$SP = \langle \mathcal{J}, \mathcal{T}, \mathcal{R}, t_f \rangle,$$

where t_f is the desired cycle time which could be either constant or variable.

As for the cost function, the total energy consumption which is to be minimized is given by

$$E = \sum_{J\in\mathcal{J}} \sum_{i=1}^{h_J} \sum_{k=p_{min}}^{p_{max}} O^{P,k}_{J,i} (O^f_{J,i} - O^s_{J,i})^k, \qquad (6)$$

which expresses the sum over all the evaluated cost polynomial, in all operations, in all jobs. The constraints describing execution are now expressed as

$$\begin{gathered} O^J_s + O^J_{min} \leq O^J_f, \\ O^J_f - O^J_s \leq O^J_{max}, \\ \forall \{O^J : O^J \in \mathcal{O}^j\}, \{J : J \in \mathcal{J}\}. \end{gathered} \qquad (7)$$

Also, since we are minimizing energy levels, we must assume that in each job, one operation is executing at all time. Idle time is modeled by idle operations. We do this by

$$\sum_{O^J \in \mathcal{O}^J} \left(O^J_f - O^J_s\right) = t_f, \qquad \forall \{J : J \in \mathcal{J}\}. \qquad (8)$$

That is, the sum of all operations in a job should amount to the desired cycle time. We should also ensure that no operation executes outside our cycle time

$$O^J_f \leq t_f, \qquad \forall \{O^J : O^J \in \mathcal{O}^j\}, \{J : J \in \mathcal{J}\}. \qquad (9)$$

Note that to avoid redundant constraints, inference can be used to skip adding the above constraint for operations which may never execute last.

As for the temporal orderings:

$$T^L \leq T^R, \qquad \forall \{T : T \in \mathcal{T},\ T_{type} = leq\} \qquad (10)$$
$$T^L = T^R, \qquad \forall \{T : T \in \mathcal{T},\ T_{type} = eq\} \qquad (11)$$

This implies that if example we would like to describe that $O_{J,1}$ should end before $O_{J,2}$ can start, then $T^L = O^f_{J,1}$, $T^R = O^s_{J,2}$ and $T_{type} = leq$.

For the modeling of resources, we need to generate the mutual exclusion for each possible combination of pairs. For each pair, there must also be a boolean describing which one executes first, we define a bijective mapping $\delta : \mathcal{P}_R \times \mathcal{P}_R \to D$, where D is a set of boolean variables. If the boolean $\delta(P_{R,1}, P_{R,2})$ is true, then this implies that $P_{R,1}$ executes before $P_{R,2}$. Thus, $\delta(P_{R,1}, P_{R,2}) \neq \delta(P_{R,2}, P_{R,1})$. The resource constraints are given by

$$\begin{gathered} P^d_{R,l} \leq P^a_{R,k} + M(1 - \delta(P_{R,l}, P_{R,k})) \\ \forall \ \{k : k \in [1, ..., h_R] \backslash l\}, \\ \{l : l \in [1, ..., h_R]\}, \\ \{R \ : R \in \mathcal{R}\}, \end{gathered} \qquad (12)$$

where M is a constant sufficiently large to nullify the constraint if the boolean is false. Note that, if for example a resource is allocated at the start of $O_{J,1}$ and deallocated at the end of $O_{J,2}$, then the pair $P_{R,l}$ describing this would have $P^a_{R,l} = O^s_{J,1}$ and $P^d_{R,l} = O^f_{J,2}$.

Note that each job itself should be subject to mutual exclusion. Thus for each job we add a corresponding resource which all operations in the job books during execution. Also, to simplify the model, if it is known that a number of operations are to execute in sequence, for example $O_{J,1}, ..., O_{J,5}$. Then it is enough to define a booking from $O^s_{J,1}$ to $O^f_{J,5}$.

V. Case Study

The robot and their operations were modeled using an offline programming and simulation environment for robot systems, ABB Robot Studio [29]. The software automatically generates time optimal trajectories for each operation. The path information for each operation can then be exported into MATLAB where the DP algorithm described in Section III was implemented.

TABLE I
TEST SET STATISTICS

	Problem instances			
	A	B	C	D
Variables	274	374	415	415
Binary	54	108	99	99
Nonlinear	106	130	154	154
Constraints	389	523	601	585
Equalities	94	109	128	128
Inequalities	295	414	473	457
Nonzeros	220	266	316	316
Robots	4	3	4	4
Zones	4	3	6	6
Operations	110	133	158	158
Sequences	16	18	24	24
Minimum time	73 s	95 s	100 s	113 s

The case study consists of four different problem instances. Each of similar size but with varied characteristics. A problem instance consists of a number of six-joint robots, operations and common zones. The operations are grouped into a number of sequences. Each sequence consists of 1-5 operations. The following provides a brief description.

(Instance A) Four robots each have four sequences. Each of these sequences are to be performed in one of four common zones. However, the first two sequences, as well as last two can be performed in an arbitrary order.

(Instance B) Three robots each perform six sequences in an arbitrary order. Each robot has two adjacent common zones (in total three) in which the sequences are performed.

(Instance C) Four robots each perform six sequences. The six sequences form three pairs, which should be performed in order, while the two sequences in each pair can be performed in any order.

(Instance D) In this job shop type instance, four robots perform six sequences in a row, each in one of six common zones.

See Table I for a compilation of statistics for the four problem instances. Note that the number of binary variables may not necessarily reflect the number of feasible alternatives for a specific problem instance.

All optimization was run on a Windows 7 64bit system with a 2.66 [GHz] Intel Core2 Quad CPU and 4 [GB] of RAM. The minimum energy trajectory planning problem for each operation instance was solved in close to 40 [s]. All MINLP benchmarks were performed using Bonmin (v1.5.1) with Cbc (v2.7.5) as MILP solver and Ipopt (v3.10.1) as NLP solver.

For each instance, 11 MINLP formulations were created with a range of final times. The resulting algorithm benchmarks are compiled in Table II. Note that the DP preprocessing time has been excluded and as such, the solution times refer only to the time taken by the MINLP scheduling algorithm. As can be seen, both NLP-BB and OA display quite long solution times for instance A and D. They are also unable to solve any of the problems in instance B and C. The two other algorithms, LP/NLP-BB and ECP manage better, both in terms of solution time as well as solved problems. Time wise, ECP seem to solve simple problems slightly faster than LP/NLP-BB. Solution wise, LP/NLP-BB works better for the larger instances. It is the only algorithm to solve all problems within 1000 [s] with a termination gap of 1%.

Instances B and C were shown to be the hardest to solve. The first is characterized by a high flexibility in the order of which the sequences are performed. The second has a medium amount of flexibility but a high number of common zones. Instance D has a high a number of common zones as C, but with its job shop style ordering of sequences, optimal solutions could be found quite fast.

The resulting overall minimum energy consumption for each instance as a function of cycle time is illustrated in Figure 2. Scheduling based on the Dynamic Programming method (dynamic scaling) is indicated by the solid curves.

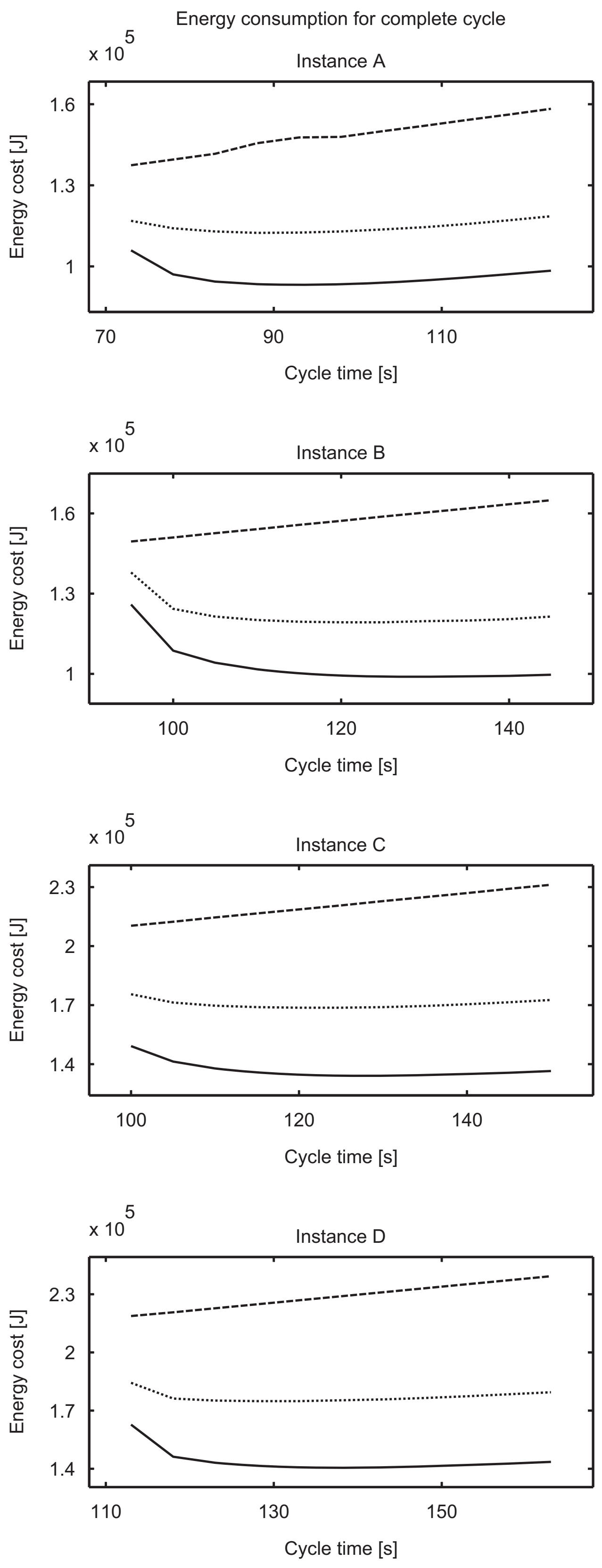

Fig. 2. Overall minimum energy consumption for the case study as a function of cycle time. The dashed curves shows scheduling where no scaling is allowed, the dotted show the result from linearly scaled operation ($\ddot{\tau} = 0$) while the solid show dynamically scaled operations as described in Section III.

TABLE II

TEST SET RESULTS

Solved instances: 0 % termination gap, 1000 s timeout

	A	B	C	D
NLP-BB	100 %	0 %	0 %	100 %
OA	100 %	0 %	0 %	82 %
LP/NLP-BB	100 %	45 %	82 %	100 %
ECP	100 %	18 %	73 %	100 %

Mean solution time: 0 % termination gap, 1000 s timeout

	A	B	C	D
NLP-BB	461 s	-	-	339 s
OA	238 s	-	-	518 s
LP/NLP-BB	39 s	462 s	335 s	70 s
ECP	27 s	282 s	369 s	55 s

Solved instances: 1 % termination gap, 1000 s timeout

	A	B	C	D
NLP-BB	100 %	0 %	0 %	100 %
OA	100 %	0 %	0 %	82 %
LP/NLP-BB	100 %	100 %	100 %	100 %
ECP	100 %	91 %	82 %	100 %

Mean solution time: 1 % termination gap, 1000 s timeout

	A	B	C	D
NLP-BB	406 s	-	-	296 s
OA	236 s	-	-	507 s
LP/NLP-BB	38 s	162 s	321 s	46 s
ECP	26 s	435 s	252 s	42 s

(Problem instances: A, B, C, D)

For comparison, we have also plotted the result of scheduling based on time optimal individual operations (dashed curves) as well as linearly scaled operations ($\ddot{\tau} = 0$, dotted curves). Compared to a system optimized using time optimal robot movements, all instances show a significant decrease in energy consumption for scheduling based on both linear and dynamic scaling.

VI. Discussion and Conclusion

In this paper we have examined scheduling problems with nonlinear energy cost functions. We have benchmarked four different methods for solving the MINLP: Nonlinear Programming based Branch and Bound, Outer Approximation, LP/NLP based Branch and Bound (LP/NLP-BB) and Extended Cutting Plane (ECP). We show that LP/NLP-BB performs best on all instances, with ECP as a close second.

For systems where the sequence of operations is given beforehand and scheduling only with regards to common zones has to be performed, the scheduling problem can be solved readily. A practical implementation of the DP method used for preprocessing is still needed to verify the data which the MINLP is based on. However, if DP results from a real process in anywhere near the computational results, the possibilities for energy reduction by MINLP scheduling are very good.

References

[1] R. Saidur. A review on electrical motors energy use and energy savings. *Renewable and Sustainable Energy Reviews*, 14(3):877 – 898, 2010.

[2] R. Visinka. *Ch. 2 - Energy Efficent Three-Phase AC Motor Drives for Appliance and Industrial Applications*. Goldberg and Middleton, 2002.

[3] G. Hirzinger, N. Sporer, A. Albu-Schaffer, M. Hahnle, R. Krenn, A. Pascucci, and M. Schedl. Dlr's torque-controlled light weight robot iii-are we reaching the technological limits now? In *Robotics and Automation, 2002. IEEE International Conference on*, 2002.

[4] J. S. Park. Motion profile planning of repetitive point-to-point control for maximum energy conversion efficiency under acceleration conditions. *Mechatronics*, 6(6):649 – 663, 1996.

[5] E.S. Sergaki, G.S. Stavrakakis, and A.D. Pouliezos. Optimal robot speed trajectory by minimization of the actuator motor electromechanical losses. *J. Intell. Robotics Syst.*, 33:187–207, Feb. 2002.

[6] O. Maimon, E. Profeta, and S. Singer. Energy analysis of robot task motions. *Journal of Intelligent and Robotic Systems*, 4:175–198, 1991.

[7] T. Izumi, H. Zhou, and Z. Li. Optimal design of gear ratios and offset for energy conservation of an articulated manipulator. *Automation Science and Engineering, IEEE Transactions on*, 6(3):551 –557, Jul. 2009.

[8] A. Kobetski and M. Fabian. Velocity balancing in flexible manufacturing systems. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 358 –363, may 2008.

[9] O. Wigström, B. Lennartson, A. Vergnano, and C. Breitholtz. High level scheduling of energy optimal trajectories. *IEEE Transactions on Automation Science and Engineering*, 2013.

[10] J. N. Hooker. Logic-based modeling, 2002.

[11] O. Wigström, N Sundström, and B Lennartson. Optimization of hybrid systems with known paths. In *Analysis and Design of Hybrid Systems (ADHS), 4th IFAC Conference on*, 2012.

[12] Pierre Bonami, Mustafa Kilinç, and Jeff Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 1–39. Springer New York, 2012.

[13] Pierre Bonami, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, and Nicolas Sawaya. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, May 2008.

[14] A. H. Land and A. G Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[15] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, January 1965.

[16] Omprakash K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546, 1985.

[17] Marco Duran and Ignacio Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[18] Roger Fletcher and Sven Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

[19] I. Quesada and I.E. Grossmann. An lp/nlp based branch and bound algorithm for convex minlp optimization problems. *Computers and Chemical Engineering*, 16(10-11):937 – 947, 1992.

[20] Tapio Westerlund and Frank Pettersson. An extended cutting plane method for solving convex minlp problems. *Computers and Chemical Engineering*, 19, Supplement 1(0):131 – 136, 1995.

[21] J.N. Hooker and M.A. Osorio. Mixed logical/linear programming. *Discrete Applied Mathematics*, 96:96–97, 1997.

[22] A. Gasparetto and V. Zanotto. A technique for time-jerk optimal planning of robot trajectories. *Robotics and Computer-Integrated Manufacturing*, 24(3):415 – 426, 2008.

[23] Alan S. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):pp. 219–223, 1960.

[24] H.M. Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959.

[25] E.H. Bowman. The schedule-sequencing problem. *Operations Research*, 7(5):pp. 621–624, 1959.

[26] P. Fattahi, M.S. Mehrabad, and F. Jolai. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 18:331–342, 2007.

[27] Cemal ˙ Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6):1539 – 1548, 2010.

[28] O. Wigström and B Lennartson. Scheduling model for systems with complex alternative behaviour. In *Automation Science and Engineering (CASE), 2012 IEEE Conference on*, 2012.

[29] "ABB RobotStudio" Internet: http://www.robotstudio.com, [Aug. 1, 2011].