Elektrotehniški vestnik 70(5): 310–316, 2003
Electrotechnical Review, Ljubljana, Slovenija

Učinkovita okolica za optimiranje tehnoloških procesov s tehniko lokalnega iskanja

Boštjan Murovec

Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška 25, 1000 Ljubljana, Slovenija
E-pošta: bostjan@lie.fe.uni-lj.si

Povzetek. Tehnika lokalnega iskanja je v zadnjem času postala priljubljeno orodje za reševanje diskretnih kombinatoričnih optimizacijskih problemov. Zasnovana je na iterativnem izboljševanju začetnega rezultata, kjer se v vsakem koraku optimizacije izvede majhna sprememba (perturbacija) nad trenutno rešitvijo. Problemi, ki jih rešujemo z lokalnim iskanjem, so največkrat prezahtevni za ekzaktno optimiranje, s katerim naj bi dobili teoretično najboljšo rešitev. V takih primerih se zadovoljimo z aproksimativnim reševanjem, kjer se optimalni rešitvi samo bolj ali manj približamo. Za doseganje zadovoljivih blizuoptimalnih rešitev je ključnega pomena izbira sprememb, ki jih nad rešitvijo izvajamo med optimizacijo. Pametna izbira le-teh dostikrat pripelje do zelo kakovostnih rešitev. Ključni korak pri zasnovi algoritma lokalnega iskanja je torej definicija ustrezne okolice, t.j. množice sprememb, ki jih nad trenutno rešitvijo lahko izvajamo. V primeru optimizacije tehnoloških pocesov, kjer je rešitev problema urnik izvajanja operacij s čim krajšim izvršnim časom, je kvalitetno okolico težko definirati, saj so lastnosti, ki jih od nje pričakujemo, protislovne. Različni avtorji so predlagali različne okolice, ki so vsaka na svoj način skušale uravnotežiti nasprotujoče si zahteve. V tem delu predlagamo novo okolico, s katero nam je uspelo odpraviti enega od kompromisov, ki uspešnost lokalnega iskanja omejujejo. Empirični rezultati so spodbudni, saj smo s svojo okolico dosegli rekordne rešitve na treh standardnih testnih primerih optimizacije tehnoloških procesov.

Ključne besede: optimizacija tehnoloških procesov, kombinatorične optimizacije, diskretne optimizacije, lokalno iskanje, hevristične optimizacije

An effective neighborhood function for the job-shop scheduling by the local search

Extended abstract. The job-shop [6] is a well known NP-hard discrete combinatorial optimization problem [7]. Because of its unmanageable complexity it is generally not possible to solve it optimally. Therefore, we are forced to seek sub-optimal solutions using one of the various approximation algorithms [16]. Among these the local search has emerged as a very successful technique [20]. It is an iterative method where schedules are optimized by performing a number of small moves (perturbations) on them.

Since the set of all possible moves that can be applied to an optimizing (current) solution is astonishingly large, it is necessary to introduce the so called neighborhood function. The neighborhood specifies the subset of moves that are considered as potentially viable to optimization. The success of the local search depends largely on the definition of the neighborhood function [11].

In case of the job-shop problem there are five properties that characterize a good neighborhood function [15]. These are: correlation, feasibility, improvement, size and connectivity. However, specified properties contradict each other, thus causing the necessity to seek a compromise among them and preventing the possibility to define the theoretically perfect neighborhood. Several authors have proposed neighborhood functions by emphasizing different sets of stated properties as essential [2, 5, 14, 18, 12].

This papers proposes yet another neighborhood definition that utilizes the recently discovered repairing technique for the job-shop problem [17]. In this way at least one important compromise can be avoided, which makes our neighborhood function a serious candidate for the replacement of similar functions proposed so far. Empirical results are encouraging, since we have got new results on three well-tested and long-known standard benchmark problem instances.

Key words: job-shop optimization, combinatorial optimization, discrete optimization, local search, heuristic optimization

Prejet 19. avgust, 2002
Odobren 24. marec, 2003

1 Uvod

Problem optimizacije tehnoloških procesov (ang. *job-shop scheduling problem*) je definiran tako, kot je opisano [6]. Podano je končno število n opravil $\{\mathcal{J}_i\}_{i=1}^{n}$, ki morajo biti izvedena s pomočjo končnega števila m strojev $\{\mathcal{M}_j\}_{j=1}^{m}$. Vsako opravilo $\mathcal{J}_i$ je zaporedje $n(i)$ operacij $w_{i,1}, \ldots w_{i,n(i)}$, od katerih mora biti vsaka izvedena na predpisanem stroju $\mathcal{M}_{i,k} \in \mathcal{M}$, za kar potrebuje predpisan čas izvajanja $p_{i,k} > 0$. Operacije, ki pripadajo istemu opravilu, so izpostavljene tehnološkim omejitvam: začetni čas izvajanja $t_{i,k}$ operacije $w_{i,k}$, ki ni

prva v opravilu ($k > 1$), ne sme biti krajši od končnega časa izvajanja $e_{i,k-1} = t_{i,k-1} + p_{i,k-1}$ njene tehnološke predhodnice $w_{i,k-1}$; v primeru prve operacije mora veljati $t_{i,1} \geq 0$ za vsak $i = 1 \ldots n$. Vsak stroj lahko izvaja samo eno operacijo hkrati, kar pomeni zmogljivostne omejitve. Poleg tega prekinjanje izvajanja operacij ni dovoljeno. Izvršni čas urnika $C_{\max}$ je dolžina časovnega intervala, v katerem se izvedejo vsa opravila. Poiskati želimo urnik, katerega izvršni čas bo čim krajši.

Zaradi velike kompleksnosti problema, ki spada v množico NP-kompletnih diskretnih kombinatoričnih optimizacijskih problemov [13], so ekzaktni algoritmi reševanja, ki jamičijo optimalnost dobljene rešitve, tako rekoč neuporabni. Namesto njih se poslužujemo hevrističnih algoritmov, ki optimalnosti rešitev ne zagotavljajo, s čimer se njihova kompleksnost zmanjša na sprejemljivo raven. Popularni pristopi reševanja so prioritetna pravila (izredno hitra, rešitve so slabe [4]), postopno ohlajanje (počasna konvergenca [12]), genetski algoritmi (slaba konvergenca k blizu-optimalnim rešitvam [8]) in lokalno iskanje, ki velja za zelo perspektiven pristop [20].

2 Predstavitev z grafom

Za opis algoritmov se instance obravnavanega problema prikazujejo s pomočjo disjunktivnega vozliščno-uteženega grafa $\mathcal{G}' = \{\mathcal{N}, \mathcal{A} \cup \mathcal{E}\}$, kjer $\mathcal{N}$, $\mathcal{A}$ in $\mathcal{E}$ označujejo množico vozlišč, množico usmerjenih povezav in množico neusmerjenih povezav [19]. Vsaka operacija $w_{i,j}$ je predstavljena s svojim vozliščem, katerega utež je enaka času izvajanja operacije $p_{i,j}$. Poleg vozlišč za vse operacije v instanci vsebuje graf dve dodatni vozlišči za fiktivni operaciji izvor $\circledcirc$ in ponor $\circledast$, katerih utež je enaka nič; z njima predstavimo začetek in konec izvajanja urnika. Za vse operacije, ki niso prve v opravilu $w_{i,j>1}$, vsebuje množica $\mathcal{A}$ povezavo $w_{i,j-1} \xrightarrow{\mathcal{A}} w_{i,j}$. V tej množici so še povezave $\circledcirc \xrightarrow{\mathcal{A}} w_{i,1} \; \forall \; (1 \leq i \leq n)$ in $w_{i,n(i)} \xrightarrow{\mathcal{A}} \circledast \; \forall \; (1 \leq i \leq n)$. Povezave v množici $\mathcal{A}$ ponazarjajo tehnološke omejitve med operacijami: povezava $w_a \xrightarrow{\mathcal{A}} w_b$ zahteva, da se mora w_a izvesti v celoti, preden lahko pričnemo izvajati w_b.

Zmogljivostne omejitve strojev podaja množica $\mathcal{E}$. Med vsakima operacijama w_a in w_b, ki se morata izvesti na istem stroju, vpeljemo neusmerjeno povezavo $w_a \xleftrightarrow{\mathcal{E}} w_b$. Povezave v množici $\mathcal{E}$ niso usmerjene, ker ob specifikaciji instance ni določeno, v kakšnem zaporedju morajo stroji izvajati operacije. To je znano šele ob določitvi urnika, po katerem bomo operacije izvedli: definirati urnik pomeni izbrati smer vsake od neusmerjenih povezav na tak način, da je rezultirajoči graf $\mathcal{G} = \{\mathcal{N}, \mathcal{A} \cup \mathcal{E}\}$ acikličen [15].

Izvršni čas urnika $C_{\max}$ je enak najdaljši uteženi poti med $\circledcirc$ in $\circledast$. Tej poti pravimo kritična pot; operacije, skozi katere poteka, imenujemo kritične operacije. Vpeljimo še pojem kritičnega bloka, ki je maksimalno zaporedje kritičnih operacij, ki se izvajajo na istem stroju. Optimizacijski algoritem mora poiskati tako konfiguracijo povezav v množici $\mathcal{E}$, da bo rezultirajoči graf imel čim krajšo kritično pot, s čimer bo pripadajoči urnik imel ustrezno kratek izvršni čas.

3 Lokalno iskanje

Lokalno iskanje je iterativna metoda optimizacije: začetni urnik se izboljšuje z izvedbo velikega števila majhnih premikov (perturbacij), ki so spremembe zaporedja izvajanja operacij na določenem stroju (kar se v grafu $\mathcal{G}$ odraža v spremembi usmeritev ustreznih povezav v množici $\mathcal{E}$); v vsakem koraku optimizacije se izvede natančno en premik, katerega rezultat je nov urnik, ki vstopa v naslednji korak optimizacije. Pri tem je pomembno, da sprememba usmeritev v množici $\mathcal{E}$ ohrani acikličnost grafa, saj je v nasprotnem primeru rezultat premika neizvedljiv urnik [15].

Že pri relativno majhnih instancah problema je vseh mogočih premikov na urniku neobvladljivo veliko (vendar končno), zato pri realizaciji lokalnega iskanja vpeljemo pojem okolice urnika, ki vsebuje samo majhno podmnožico mogočih premikov na urniku; to so premiki, za katere štejemo, da imajo veliko možnost zmanjšati izvršni čas urnika.

Idealna okolica, kateri se lahko samo bolj ali manj približamo, ima naslednje med seboj protislovne lastnosti [15]:

koreliranost- originalna in nova rešitev se ne smeta preveč razlikovati, da lahko podrobno pregledujemo prostor rešitev,

izvedljivost- rezultat perturbacije naj bo vedno izvedljiva rešitev (rezultirajoči graf acikličen),

napredovanje- okolica naj vsebuje samo tiste poteze, ki z veliko verjetnostjo vodijo k izboljšanju trenutne rešitve,

velikost- okolica naj bo čim manjša, da pregledovanje njenih rešitev ne porabi veliko časa,

povezljivost- okolica mora vsebovati končno število potez, s pomočjo katerih lahko pridemo h globalnemu optimumu od katerekoli začetne rešitve.

Pri definiciji okolice moramo narediti kompromis: velike okolice niso praktične, ker postopek lokalnega iskanja izgubi pregled nad dogajanjem, poleg tega je vrednotenje vseh potez časovno zamudno opravilo; slabost majhnih okolic pa je večja možnost, da se potencialno dobre poteze v njih ne nahajajo. Uspeh lokalnega iskanja je močno odvisen od definicije okolice [11].

4 Izhodišče za definicijo okolice

Pri definiciji okolice, ki naj bo podlaga za uspešno reševanje kateregakoli kombinatoričnega optimizacijskega problema, je nujno upoštevati teoretične izsledke, ki veljajo za obravnavani problem. Pri razvrščanju proizvodnih procesov, ki so izpostavljeni tehnološkim omejitvam, so najpomembnejše naslednje ugotovitve.

Teorem 1 *Dan imamo poljuben veljaven urnik izvajanja opravil $\mathcal{G}(1)$ s izvršnim časom $C_{\max}(1)$. V kolikor obstaja urnik $\mathcal{G}(2)$ s krajšim izvršnim časom $C_{\max}(2)$, velja vsaj ena od naslednjih trditev: (1) vsaj ena neprva operacija kateregakoli kritičnega bloka v urniku $\mathcal{G}(1)$ se mora v urniku $\mathcal{G}(2)$ izvesti pred vsemi operacijami, ki v $\mathcal{G}(1)$ sestavljajo pripadajoči blok; (2) vsaj ena nezadnja operacija kateregakoli kritičnega bloka v urniku $\mathcal{G}(1)$ se mora v urniku $\mathcal{G}(2)$ izvesti za vsemi operacijami, ki v $\mathcal{G}(1)$ sestavljajo pripadajoči blok.*

Dokaz teorema, ki pomeni fundamentalno ugotovitev za zasnovo lokalnega iskanja obravnavanega problema, najdemo v [3]. Na kratko ga lahko razložimo takole. Ker je izvršni čas $C_{\max}(2)$ krajši od $C_{\max}(1)$, ima graf urnika $\mathcal{G}(2)$ krajšo kritično pot kot graf urnika $\mathcal{G}(1)$. Kritični bloki, ki sestavljajo kritično pot, vsebujejo operacije, ki se izvajajo na istem stroju. Zato med vsako nezadnjo operacijo bloka w_x in njeno naslednico w_y obstaja povezava $w_x \overset{\mathcal{E}}{\longleftrightarrow} w_y$. Med zadnjo operacijo w_z nezadnjega kritičnega bloka in prvo operacijo w_p naslednjega bloka obstaja povezava $w_z \overset{\mathcal{A}}{\longrightarrow} w_p$, ker obe operaciji pripadata istemu opravilu (sicer operacija w_p ne bi čakala na dokončanje izvajanja operacije w_z in kritična pot ne bi potekala skozi njiju). Ker povezav v množici $\mathcal{A}$ ne moremo spremeniti, saj so dane s tehnološkimi omejitvami, lahko kritično pot zmanjšamo samo tako, da kritične operacije pomikamo izven kritičnih blokov, s čimer potencialno krajšamo pripadajoči odsek kritične poti.

Teorem 2 *Dan imamo poljuben veljaven urnik izvajanja opravil $\mathcal{G}(1)$ z izvršnim časom $C_{\max}(1)$. Če v njem spremenimo zaporedje izvajanja operacij in s tem dobimo urnik $\mathcal{G}(2)$ tako, da se spremeni samo začetna operacija prvega kritičnega bloka in/ali zadnja operacija zadnjega kritičnega bloka v $\mathcal{G}(1)$, potem velja $C_{\max}(2) \geq C_{\max}(1)$.*

Dokaz teorema najdemo v [18]. Trditev je dopolnilo prvega teorema, saj množico operacij, ki potencialno manjšajo izvršni čas urnika, še naprej skrči. Ugotovitev temelji na dejstvu, da se izvajanje prve operacije prvega kritičnega bloka ne more začeti pred časom 0, zato kakršnakoli zamenjava začetne operacije bloka ne more pospešiti izvajanja operacij v njem. Odsek kritične poti, ki pripada temu bloku, je zato vedno enak vsoti izvršnih časov operacij, ki sestavljajo blok. Za zadnji kritični blok je sklep analogen.

Teorem 3 *Če v poljubnem izvedljivem urniku zamenjamo vrstni red izvajanja dveh sosednjih kritičnih operacij v kritičnem bloku, je rezultat spemembe zopet izvedljiv urnik. Če operaciji, katerih vrstni red zamenjamo, nista sosednji na kritični poti, je mogoče, da bo rezultat zamenjave neizvedljiv urnik.*

Trditev je dokazana v [1]. Podrobnejša razlaga neizvedljivosti in primer njenega nastanka sta podana v [17].

5 Dosedanje okolice

Najbolj znane in teoretično pomembne so naslednje okolice, ki so se formirale skladno s spoznanji in razvojem teorije obravnavanega problema.

Van Laarhoven in sod. [12] okolica $\mathcal{H}_L$ vsebuje vse zamenjave zaporedja dveh sosednjih operacij na stroju, če se obe nahajata na kritični poti. Smisel take definicije je v tem, da zamenjava dveh nekritičnih operacij ne more zmanjšati izvršnega časa, saj dobljeni graf še vedno vsebuje prvotno kritično pot. Poleg tega je rezultat vseh njenih perturbacij izvedljiv urnik [1]. Dokazano je, da ima ta okolica lastnost povezljivosti. Njena slaba stran pa je, da je prevelika in da vsebuje poteze, ki same zase ne vodijo k izboljšanju izvršnega časa urnika (teorem 1).

Matsuo in sod. [14] okolica $\mathcal{H}_M$ vsebuje vse zamenjave dveh sosednjih operacij na stroju, ki se nahajata na kritični poti, s tem da se vsaj ena od njih nahaja na robu bloka kritične poti (izbira sledi iz teorema 1). Ta okolica je občutno manjša od prejšnje (velja relacija $\mathcal{H}_M \subseteq \mathcal{H}_L$), je pa zato izgubila lastnost povezljivosti.

Nowicki in Smutnicki [18] reducirata okolico $\mathcal{H}_M$ s tem, da dokažeta, da sprememba začetne operacije začetnega bloka in simetrično končne operacije končnega bloka kritične poti ne more reducirati kriterijske funkcije (če se ob tem ne spremeni konec začetnega bloka oziroma začetek končnega bloka; teorem 2). Okolica $\mathcal{H}_N$ torej iz okolice $\mathcal{H}_M$ odstrani dve zamenjavi, ki teorema ne upoštevata. Velika prednost te okolice je, da je izredno majhna in zato učinkovita, saj njena nadaljnja krčitev ni možna brez izločanja premikov, ki so za optimizacijo obetavni [11]. Njena slabost pa je nepovezljivost, saj velja $\mathcal{H}_N \subseteq \mathcal{H}_M$.

Dell'Amico in Trubian [5] definirata več okolic. Na tem mestu opišimo samo tisto, ki je zanimiva za naše

raziskave ($\mathcal{H}_D$). V njej se nahajajo vse perturbacije, kjer katerokoli notranjo operacijo bloka na kritični poti premaknemo na začetek ali konec ustreznega bloka, kar v celoti ustreza teoremu 1. Neprijetna lastnost take izbire so potencialno neizvedljive poteze [17]. Če bi bil rezultirajoči urnik neizvedljiv, se premik izvede proti začetku ali koncu bloka do tistega mesta, kjer je dobljeni urnik še vedno izvedljiv. Smisel premikanja notranjih operacij zunaj blokov je torej v tem, da je to edini način zmanjševanja izvršnega časa urnika (teorem 1). Pomen premikanja operacij, ki vodijo v neizvršljive rešitve na pozicijo, kjer se legalnost še ohrani, pa je v tem, da na novo dobljena kritična pot poteka skozi predhodnico ali naslednico operacije znotraj opravila, s čimer se odpira možnost odprave ciklov v grafu pri premiku le-te zunaj svojega bloka. Tako je dosežena lastnost povezljivosti, hkrati pa se ohrani občutno večji odstotek potez, ki so potencialno napredujoče (v primerjavi z okolico $\mathcal{H}_L$, ki ima tudi lastnost povezljivosti).

Balas [2] okolica $\mathcal{H}_V$ je konceptualno podobna okolici $\mathcal{H}_D$. Od nje se razlikuje le po tem, da so potencialno nelegalne poteze zavržene. Velja torej $\mathcal{H}_V \subseteq \mathcal{H}_D$, s čimer je izgubljena povezljivost.

Postopki lokalnega iskanja najpogosteje implementirajo okolico $\mathcal{H}_N$, ker je majhna. Perspektivna okolica $\mathcal{H}_D$ se ne uporabljana veliko. Razlog je verjetno v tem, da je nelegalne poteze, ki jih legaliziramo z omejitvijo premika, zelo težko izkoristiti; postopek lokalnega iskanja bi moral biti precej inteligenten, da bi pravilno nadaljeval premik omejevane poteze s sinergičnim premikom tehnoloških predhodnic ali naslednic premaknjene operacije.

Slabost lahko odpravimo z uporabo popravljalne tehnike [17], ki na ravni izvedbe premika poskrbi za sinergično prerazporeditev vseh operacij, ki preprečujejo legalni premik ustrezne operacije zunaj kritičnega bloka, v katerem se nahaja. To je ključna ugotovitev, ki ločuje našo okolico od do sedaj znanih.

6 Predlagana okolica

Cilj lokalnega iskanja je manjšanje kritične poti grafa, ki pomeni urnik, zato je smiselno v okolico vključiti samo tiste poteze, ki kritično pot kakorkoli spremenijo. Veljavnost teorema 1 je zadosten razlog, da iz prejšnje množice izločimo poteze, ki ne spremenijo začetka ali konca nobenega kritičnega bloka. S pomočjo teorema 2 izločimo poteze, ki spremenijo samo začetek prvega ali konec zadnjega kritičnega bloka. Če vztrajamo pri potezah, ki izvedljivost urnika inherentno ohranjajo, je s temi napotki mogoče zgraditi samo okolico $\mathcal{H}_N$, katere slaba stran je, da nima lastnosti povezljivosti.

Okolica $\mathcal{H}_N$ je torej dobro izhodišče za definicijo naše okolice, v katero bomo dodali poteze, s katerimi ji bomo povrnili povezljivost. Rešitev v smislu dodajanja notranjih potez bloka, kot je to storjeno pri okolici $\mathcal{H}_L$, ni učinkovita. Poteze, ki prerazporejajo izvajanje operacij znotraj blokov, so sicer dolgoročno lahko koristne, vendar njihovo prednost zelo težko izkoristimo. Postopki lokalnega iskanja delujejo tako, da vse poteze v okolici ovrednotijo na podlagi dejanske ali ocenjene vrednosti rezultirajočega izvršnega časa, nakar se odločijo za najboljšo potezo. Ker notranje poteze izvršnega časa ne morejo zmanjšati same zase, bodo prišle na vrsto šele, ko bodo vse druge možnosti izčrpane (npr. pri iskanju s tabu seznamom [9], ko bodo vse druge poteze prepovedane ali dale rezultat, slabši od trenutnega).

Konceptualno boljšo rešitev ponuja okolica $\mathcal{H}_D$, kjer skušamo z vsako potezo spremeniti začetek ali konec vsaj enega kritičnega bloka. Okolica $\mathcal{H}_C$, ki jo predlagamo, je vsebinsko podobna okolici $\mathcal{H}_D$. Edina razlika je v tem, da nelegalnih potez zaradi uporabe popravljalne tehnike ne obravnavamo na poseben način [17]. Namesto da premikamo operacije znotraj bloka proti zunanjosti do najbolj oddaljene pozicije, kjer je acikličnost grafa še ohranjena, jih vedno premaknemo na njihovo končno pozicijo, kar je v okolici $\mathcal{H}_D$ cilj, do katerega se s sedanjimi postopki nismo mogli dokopati neposredno.

Definicija okolice je naslednja. Kritična pot vsebuje končno zaporedje blokov: $KP = B_1, \ldots, B_i, \ldots, B_Q$. Vsak blok vsebuje končno zaporedje operacij, ki se izvajajo na istem stroju: $B_i = b_1, \ldots, b_j, \ldots, b_k$. Če blok ni niti prvi niti zadnji na kritični poti, dodamo v okolico $\mathcal{H}_C$ vse premike neprvih operacij na začetek bloka in vse premike nezadnjih operacij na konec bloka. Če blok vsebuje samo eno operacijo, ne moremo izvesti nobenega premika. V prvem bloku na kritični poti premikamo samo vse nekončne operacije na konec bloka, v zadnjem pa samo neprve na začetek bloka. Formalneje lahko zapišemo okolico takole:
prvi blok na kritični poti:

$$\mathcal{H}_{CZ1} = \emptyset$$

$$\mathcal{H}_{CK1} = \begin{cases} B_1 \longrightarrow b_2, \ldots, b_k, b_a; & \\ & a = 1 \\ B_1 \longrightarrow b_1, \ldots, b_{a-1}, b_{a+1}, \ldots, b_k, b_a; & \\ & a = 2, \ldots, k-1 \end{cases}$$

od drugega do predzadnjega bloka na kritični poti:

$$\mathcal{H}_{CZi} = \begin{cases} B_i \longrightarrow b_a, b_1, \ldots, b_{a-1}, b_{a+1}, \ldots, b_k; & \\ & a = 2, \ldots, k-1 \\ B_i \longrightarrow b_a, b_1, \ldots, b_{k-1}; & \\ & a = k \end{cases}$$

$$\mathcal{H}_{CKi} = \begin{cases} B_i \longrightarrow b_2, \ldots, b_k, b_a; \\ \qquad\qquad\qquad\qquad a = 1 \\ B_i \longrightarrow b_1, \ldots, b_{a-1}, b_{a+1}, \ldots, b_k, b_a; \\ \qquad\qquad\qquad\qquad a = 2, \ldots, k-1 \end{cases}$$

zadnji blok na kritični poti:

$$\mathcal{H}_{CZQ} = \begin{cases} B_Q \longrightarrow b_a, b_1, \ldots, b_{a-1}, b_{a+1}, \ldots, b_k; \\ \qquad\qquad\qquad\qquad a = 2, \ldots, k-1 \\ B_Q \longrightarrow b_a, b_1, \ldots, b_{k-1}; \\ \qquad\qquad\qquad\qquad a = k \end{cases}$$

$\mathcal{H}_{CKQ} = \emptyset$

Oznaki $\mathcal{H}_{CZi}$ (premiki operacij b_a na začetek bloka) in $\mathcal{H}_{CKi}$ (premiki operacij b_a na konec bloka) sta podmnožici premikov, ki jih v okolico $\mathcal{H}_C$ prispeva posamezni kritični blok B_i. Celotna okolica je unija vseh teh množic:

$$\mathcal{H}_C = \bigcup_{i=1}^{Q} (\mathcal{H}_{CZi} \cup \mathcal{H}_{CKi}).$$

Okolica $\mathcal{H}_C$ ima dve zanimivi lastnosti:

1. Če za neki urnik velja $\mathcal{H}_C = \emptyset$, je le-ta optimalen (nasprotno seveda ne velja). Množica potez je namreč prazna samo, ko je vsaj en kritični stroj ali vsaj eno kritično opravilo optimalno razporejeno prek celotnega časa izvajanja urnika brez časovnih zakasnitev katerekoli operacije (vsi bloki na kritični poti vsebujejo eno operacijo, ali pa kritična pot vsebuje samo en blok),
2. Okolica $\mathcal{H}_C$ vsebuje samo poteze, ki so obetajoče za zmanjšanje kriterijske funkcije. To lastnost imata od opisanih okolic še $\mathcal{H}_N$ in $\mathcal{H}_V$, ki pa nimata lastnosti povezljivosti. Okolica $\mathcal{H}_D$, ki je povezljiva, pa vsebuje poteze, za katere je znano, da same zase ne morejo zmanjšati izvršnega časa urnika, kljub temu pa morajo biti vsebovane, da se lastnost povezljivosti ohranja (ki pa jo, kot smo že omenili, težko izkoristimo).

Kolikor vemo, je $\mathcal{H}_C$ prva okolica, ki združuje ti dve lastnosti, zato sklepamo, da nam je uspelo zaobiti vsaj en kompromis, ki je vplival na do sedanj znane okolice.

7 Empirični rezultati

Predlagano okolico smo preiskusili s tabu iskanjem (*ang. tabu search*) v kombinaciji z genetskim algoritmom [10].

Iskanje s tabu seznamom deluje nad poljubnim začetnim urnikom. Łtevilo iteracij brez izboljšave urnika je enako številu opravil v instanci. Dolžina seznama prepovedanih potez je v vsakem koraku enaka polovici števila operacij na trenutni kritični poti urnika (na začetku je seznam prazen). Tako smo skušali doseči vsaj delno adaptacijo algoritma na različne instance. V vsaki iteraciji se oceni $C_{\max}$ za vse poteze v okolici $\mathcal{H}_C$. Postopek izbere potezo z najmanjšim $C_{\max}$, ki ni na trenutnem tabu seznamu, in jo izvede. Na tabu seznam uvrstimo potezo, ki bi opravila reverzno spremembo, s čimer je le-to prepovedano izvajati toliko iteracij, kot je trenutna dolžina tabu seznama. Postopek vrne najboljšo rešitev, na katero je naletel med optimizacijo.

Genetski algoritem je deloval z relativno majhno populacijo 30 kromosomov, ker se je predhodno opisani algoritem tabu iskanja izkazal za zelo močnega, s čimer je mogoče dobiti dobre optimizacijske rezultate z manjšo populacijo, kot je to v navadi pri izvedbah podobnih algoritmov. Kriterij za končanje algoritma je bilo 15 zaporednih iteracij brez izboljšave najboljšega kromosoma. Na začetku smo z vsakim kromosom zakodirali naključno generiran legalni urnik in ga optimirali s tabu iskanjem. Kromosomom smo priredili oceno njihove uspešnosti na podlagi izvršnega časa urnika. Ocene smo skalirali [10] tako, da se je razmere med najboljšim in povprečnim kromosomom razlikovalo s faktorjem 5. Nato smo izvedli križanje tipa GOX [15] nad 10 odstotki naključno izbranih kromosomov (verjetnost izbire je bila proporcionalna oceni uspešnosti). Rezultirajoče kromosome smo optimirali s tabu iskanjem in jih uvrstili v populacijo pod pogojem, da kromosom z enakim $C_{\max}$ v populaciji še ni obstajal. Za križanjem smo izvedli mutacijo nad 30 odstotki izbranih kromosomov z verjetnostjo mutacije gena 1 odstotka. Rezultate smo obravnavali na enak način kot pri križanju. Na koncu iteracije smo z naključno izbiro (na podlagi ocene uspešnosti) izločili toliko kromosomov, da je populacija imela prvotno velikost.

Pri reševanju standardnih testnih problemov, ki so kvalificirani kot težko rešljivi [11], smo dobili rezultate, ki jih podaja tabela 1.

Opazimo, da smo na treh testnih primerih (ABZ9, YN1 in YN2) dosegli izboljšavo glede na do sedaj znane rezultate, ki se že od leta 1997 niso spremenili, čeprav so bili s temi testi preiskušeni vsi objavljeni postopki reševanja. Poudariti moramo še, da je bila populacija kromosomov zelo majhna, saj so implementacije z več sto kromosomi pogoste. Poleg tega nismo izvajali nikakršnih nastavitev parametrov individualno za posamezno instanco, kar avtorji tako rekoč vedno storijo. Razlog je v tem, da smo želeli preiskusiti, kako se predlagani optimizacijski postopek obnese pri popolnoma avtomatskem delovanju, torej brez vsakršne intervencije uporabnika.

Za nazornejši prikaz optimizacijskih sposobnosti predlagane okolice podajamo v tabeli 2 primerjalne rezul-

test	znani $C_{\max}$	dobljeni $C_{\max}$	izvršni čas (s)
ABZ7	656	658	77,75
ABZ8	665	669	104,88
ABZ9	**679**	**678**	93,39
FT10	930	930	1,15
LA21	1046	1046	3,07
LA24	935	935	3,07
LA29	1152	1153	16,30
LA40	1222	1222	22,90
YN1	**888**	**886**	220,78
YN2	**909**	**907**	242,28
YN3	893	895	228,23
YN4	968	969	263,36

Tabela 1. Rezultati standardnih testov
Table 1. Results of standard benchmark tests

tate povprečnih izvršnih časov urnikov, ki jih generira naša okolica ter tri izmed predhodno opisanih okolic, ki dosegajo najboljše rezultate. V stolpcu z oznako Δ je podana razlika med rezultati naše in primerjane okolice, pri čemer pozitivna številka pomeni, da smo z našo okolico dosegli boljše rezultate.

8 Sklep

V delu smo definirali učinkovito okolico urnika, s pomočjo katere je mogoče izvesti postopke lokalnega iskanja, ki so v tem trenutku najbolj obetavno orodje za optimizacijo tehnoloških procesov. Empirični testi pristop spodbujajo, saj smo s predlagano okolico dosegli rekordne rezultate na treh standardnih testnih instancah obravnavanega problema.

9 Literatura

[1] Egon Balas. Machine scheduling via disjunctive graphs: An implicit enumeration algorithm. *Operations Research*, 17:941–957, 1969.

[2] Egon Balas and Alkis Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44:262–275, 1998.

[3] Peter Brucker, Bernd Jurisch, and Bernd Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, 49:107–127, February 1994.

[4] Yih-Long Chang, Toshiyuki Sueyoshi, and Robert S. Sullivan. Ranking dispatching rules by data envelopment analysis in a job-shop environment. *IIE Transactions*, 28:631–642, 1996.

[5] Mauro Dell'Amico and Marco Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 1993.

[6] Simon French. *Sequencing and Scheduling: An introduction to the mathematics of the Job-Shop*. Ellis Horwood, John Willey & Sons, New York, 1982.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Franzisco, 1979.

[8] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms and Engineering Design*. John Wiley & Sons, 1997.

[9] Fred Glover. Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[10] David E. Goldberg. *Genetic Algorithms in Search, Opti-*

primerjava okolice $\mathcal{H}$ z okolico $\mathcal{H}_N$			
	povprečni $C_{\max}$		
problem	$\mathcal{H}$	$\mathcal{H}_N$	Δ
abz7	667,48	672,79	5,31
abz8	676,42	682,83	6,41
abz9	689,25	695,80	6,55
ft10	931,66	936,61	4,96
la21	1049,47	1052,29	2,82
la24	938,57	941,43	2,87
la29	1167,18	1176,58	9,40
la40	1226,56	1230,19	3,63
yn1	897,16	901,43	4,27
yn2	917,69	923,84	6,15
yn3	901,97	906,99	5,02
yn4	980,58	986,67	6,09
primerjava okolice $\mathcal{H}$ z okolico $\mathcal{H}_D$			
	povprečni $C_{\max}$		
problem	$\mathcal{H}$	$\mathcal{H}_D$	Δ
abz7	667,48	668,18	0,70
abz8	676,42	677,92	1,50
abz9	689,25	690,73	1,47
ft10	931,66	933,46	1,80
la21	1049,47	1049,97	0,50
la24	938,57	938,60	0,03
la29	1167,18	1176,60	9,42
la40	1226,56	1227,14	0,58
yn1	897,16	897,83	0,67
yn2	917,69	918,70	1,01
yn3	901,97	903,25	1,27
yn4	980,58	985,78	5,20
primerjava okolice $\mathcal{H}$ z okolico $\mathcal{H}_V$			
	povprečni $C_{\max}$		
problem	$\mathcal{H}$	$\mathcal{H}_V$	Δ
abz7	667,48	668,22	0,74
abz8	676,42	678,04	1,62
abz9	689,25	690,80	1,55
ft10	931,66	933,47	1,81
la21	1049,47	1049,97	0,50
la24	938,57	938,54	-0,03
la29	1167,18	1174,62	7,44
la40	1226,56	1226,88	0,32
yn1	897,16	897,92	0,76
yn2	917,69	919,04	1,35
yn3	901,97	903,43	1,46
yn4	980,58	986,38	5,80

Tabela 2. Primerjava povprečnih rezultatov okolice $\mathcal{H}$ z okolicami $\mathcal{H}_N$, $\mathcal{H}_D$ in $\mathcal{H}_V$
Table 2. Comparison of average results of neighborhood $\mathcal{H}$ with neighborhoods $\mathcal{H}_N$, $\mathcal{H}_D$ and $\mathcal{H}_V$

mization and Machine Learning. Addison-Wesley, 1989.

[11] Anant S. Jain. *A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem*. PhD thesis, Department of Applied Physics and Electronic and Mechanical Engineering, University of Dundee, UK, 1998.

[12] Peter J. M. Van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job-shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

[13] J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

[14] Hirofumi Matsuo, Chank Juck Suh, and Robert S. Sullivan. A controlled search simulated annealing method for the general job-shop scheduling problem. Technical report, Department of Management, Graduate School of Business, The University of Texas at Austin, Austin, TX 78712, 1988.

[15] Dirk Christian Mattfeld. *Evolutionary Search and the Job Shop: Investigations on Genetic Algorithms for Production Scheduling*. Physica-Verlag, Heidelberg, Germany, 1996.

[16] Thomas E. Morton and David W. Pentico. *Heuristic Scheduling Systems*. John Wiley & Sons, 1993.

[17] Boštjan Murovec. Netvegano izvajanje sprememb na urnikih tehnoloških procesov. *Elektrotehniški vestnik*, 68(5):313–318, 2001.

[18] Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42:797–813, 1996.

[19] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives. Note D.S. no. 9 bis, SEMA, Paris, France, Décembre, 1964.

[20] R.J.M Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job-shop scheduling by local search. *Informs Journal on computing*, 8:302–317, 1996.

Boštjan Murovec je diplomiral leta 1996, magistriral leta 1999 in doktoriral leta 2002 na Fakulteti za elektrotehniko, kjer je sedaj asistent. Predmet njegovega raziskovanja so hevristični algoritmi za optimizacijo tehnoloških procesov.